

Міністерство освіти і науки України
Державний заклад
«Луганський національний університет імені Тараса Шевченка»

Навчально-науковий інститут математики та інформаційних технологій

Кафедра інформаційних технологій та систем

Вернік Олег Володимирович

РОЗРОБКА КРОСПЛАТФОРМНОЇ 2D ГРИ ЗАСОБАМИ JAVA ТА LIBGDX

кваліфікаційна робота

**здобувача вищої освіти першого (бакалаврського) рівня
освітньої програми «Інженерія програмного забезпечення»
за спеціальністю 121 Інженерія програмного забезпечення**

Особистий підпис _____ Олег ВЕРНІК

Науковий керівник _____ Володимир ДОНЧЕНКО,
старший викладач
кафедри інформаційних технологій
та систем

Завідувач кафедри _____ Микола СЕМЕНОВ,
кандидат педагогічних наук, доцент
кафедри інформаційних технологій
та систем

Полтава – 2024

Міністерство освіти і науки України
Державний заклад „Луганський національний університет
імені Тараса Шевченка”

Інститут	Навчально-науковий інститут математики та інформаційних технологій
Кафедра, циклова комісія	Інформаційних технологій та систем
Рівень освіти	перший (бакалаврський)
Напрямок підготовки (спеціальність)	121 «Інженерія програмного забезпечення» (код, назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри ІТС
М.А. Семенов

(підпис) (ініціали, прізвище)
“ ” 2023 р.

**ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ**

Верніку Олегу Володимировичу
(прізвище, ім'я, по батькові)

**1. Тема проекту (роботи) Розробка кросплатформної 2D гри засобами
Java та LibGDX**

Керівник кваліфікаційної роботи Донченко В.Ю.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджена наказом по університету Від“ ” 2023 року №_

2. Строк подання студентом проекту (роботи)
3. Вихідні дані до роботи (проекту) у результаті виконання роботи
повинно бути розроблено кросплатформну 2D гру засобами Java та LibGDX

(визначаються кількісні або (та) якісні показники, яким повинен відповідати об'єкт розробки)

**4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно
розробити) МЕТОДОЛОГІЯ РОЗРОБКИ ГРАФІЧНИХ ІГРОВИХ ДОДАТКІВ
РОЗРОБКА КРОСПЛАТФОРМНОЇ ГРИ SIEGE ЗА ДОПОМОГОЮ
ФРЕЙМВОРКУ LIBGDX.**

(визначаються назви розділів або (та) перелік питань, які повинні увійти до тексту ПЗ)

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання „_____” _____ 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
	Вибір теми роботи, вивчення наукової літератури, затвердження теми та керівника.	До 15 жовтня	
	Аналіз літературних джерел за темою роботи. Розробка та апробація методики дослідно-експериментальної роботи. Подання структури теоретичної частини роботи та плану експериментальних досліджень.	Другий тиждень листопада (10 листопада)	
	Робота над теоретичною частиною. Подання теоретичної частини роботи для першого читання науковим керівником.	До 15 грудня	
	Усунення зауважень, урахування рекомендацій наукового керівника. Подання теоретичної частини роботи на друге читання.	До 28 січня	
	Проведення експериментальної роботи. Поетапний аналіз та обговорення її результатів. Перевірка стану виконання роботи.	Перший тиждень березня	
	Урахування рекомендацій наукового керівника, усунення недоліків, підготовка варіанта роботи до передзахисту. Розробка презентації.	До 31 березня	
	Попередній захист роботи на кафедрі	квітень	
	Доопрацювання роботи з урахуванням рекомендацій після передзахисту. Подання роботи науковому керівникові та рецензентові на підготовку відгуку та рецензії	За 10 днів до державної атестації	
	Подання на кафедру остаточного варіанта роботи, переплетеного та підписаного автором, науковим керівником і рецензентом.	За 5 днів до державної атестації	

Студент

підпис

О.В. Вернік

(ініціали, прізвище)

Керівник проекту (роботи)

підпис

В.Ю. Донченко

АНОТАЦІЯ

Вєрнік О. В.

Тема: Розробка кросплатформної 2D гри засобами Java та LibGDX.

Спеціальність: 121 "Інженерія програмного забезпечення"

Установа: ЛНУ імені Тараса Шевченка, 2024 р.

Бакалаврська робота містить: 50 с., 16 рис., 5 додат., 36 джерел.

Об'єкт дослідження – кросплатформна графічна гра.

Предмет дослідження – технологія створення кросплатформних графічних ігор засобами Java та LibGDX.

Мета роботи – розробка кросплатформної графічної 2D гри засобами Java та LibGDX.

Результати роботи. В дипломній роботі досліджено методологію розробки графічних ігрових додатків, розглянуто види комп'ютерних ігор за жанрами, кількістю гравців, стилістикою та платформами. Вивчені принципи розробки ігор та концепція кросплатформного підходу. Обрано інтегроване середовище розробки (Android Studio) та засоби, що надають можливість розробляти графічні ігри, обґрунтовано переваги застосування Java-фреймворку LibGDX для створення кросплатформних додатків, що працюють на операційних системах для ПК (Windows) та Android. Розроблено сценарій гри, класи, об'єкти та логіку їх дій.

Висновки. В результаті розробки було створено кросплатформну графічну гру «Siege» за допомогою фреймворку LibGDX.

Ключові слова. ГРАФІЧНА ГРА, ЖАНРИ ІГОР, ПРИНЦИПИ РОЗРОБКИ ІГОР, КРОСПЛАТФОРМНІСТЬ, ПК, ОС ANDROID, ANDROID STUDIO, ФРЕЙМВОРК LIBGDX.

ABSTRACT

Viernik Oleh

Theme: Development of a cross-platform 2D game using Java and LibGDX.

Speciality: 121 "Software Engineering"

Institution: Luhansk Taras Shevchenko National University (LTSNU), 2024.

Diploma work contains: 50 pages, 16 Fig., 5 adj., 36 source.

A research object is cross-platform graphics game.

The article of research is technology for creating cross-platform graphic games using Java and LibGDX.

An aim of work is development of a cross-platform graphic 2D game using Java and LibGDX.

Job performances. The thesis researched the methodology of developing graphic game applications, considered the types of computer games by genre, number of players, style and platforms. Learned principles of game development and the concept of a cross-platform approach. An integrated development environment (Android Studio) and tools that provide the ability to develop graphic games are chosen, the advantages of using the LibGDX Java framework for creating cross-platform applications that work on PC (Windows) and Android operating systems are substantiated. The game scenario, classes, objects and the logic of their actions have been developed.

Conclusions. As a result of the development, a cross-platform graphic game "Siege" was created using the LibGDX framework

Keywords. GRAPHIC GAME, GAME GENRES, PRINCIPLES OF GAME DEVELOPMENT, CROSS-PLATFORMITY, PC, ANDROID OS, ANDROID STUDIO, LIBGDX FRAMEWORK.

ИТС. ИП34.1224-ВП

Розробка кросплатформної 2D гри засобами Java та LibGDX.

[illegible]

Міністерство освіти і науки України
Державний заклад «Луганський національний університет
імені Тараса Шевченка»
Факультет (інститут) Навчально-науковий інститут математики та
інформаційних технологій

(повна назва)
Кафедра Інформаційних технологій та систем

(повна назва)

ТЕХНІЧНЕ ЗАВДАННЯ
на виконання програмної розробки (ПР) :
"РОЗРОБКА КРОСПЛАТФОРМНОЇ 2D ГРИ ЗАСОБАМИ JAVA ТА
LIBGDX"

ІТС. ІПЗ4.1224-02-ТЗ

ПОГОДЖЕНО
Керівник кваліфікаційної роботи

Донченко В. В.

“ ” 2024р.

ВИКОНАВЕЦЬ
Студент групи 4 ІПЗ

Вернік О. В.

“ ” 2024р.

Полтава 2024

ЗМІСТ

ВСТУП	3
1. ХАРАКТЕРИСТИКА ОБ'ЄКТА	3
2. ПРИЗНАЧЕННЯ ТОВАРІВ	3
3. ОСНОВНІ ВИМОГИ ДО ПРОГРАМНОГО КОМПЛЕКСУ	3
4. ТЕХНІКО - ЕКОНОМІЧНІ ВИМОГ ДО КІНЦЕВОГО ПРОДУКТУ	4
5. ВИМОГИ ДО МАТЕРІАЛІВ І КОМПЛЕКТУЮЧИХ	4
6. ЕТАПИ ВИКОНАННЯ ПР	5
7. ПРИЙОМ.....	5
8. ПОРЯДОК ВНЕСЕННЯ ЗМІН ДО ТЕХНІЧНЕ ЗАВДАННЯ, ЩО ЗАТВЕРДЖЕНО.....	6

ВСТУП

- 1.1 Найменування: Кросплатформна графічна гра.**
- 1.2 Шифр ПР: АДР-1**
- 1.3 Підстава до виконання ПР:** Підставою для виконання даної розробки є необхідність побудови геометрії двомірних та тривимірних конструкцій та автоматичної генерації розрахункових сіток для скінчено – елементного моделювання конструкцій для елементів чисельного розрахунку напружено-деформованого стану деформівного тіла.
- 1.4 Терміни розробки:**
 - 1.4.1** Початок 30 жовтня 2023 р.
 - 1.4.2** Закінчення 20 березня 2024 р.
- 1.5 Фінансується** за рахунок коштів замовника. Умови фінансування – згідно договору №12/а та протоколу погодження ціни № 12/б.

1. ХАРАКТЕРИСТИКА ОБ'ЄКТА

- 1.1.** Розробляема гра повина мати розважальний характер. **До складу об'єкту**, що створюється повинно входити:
 - 1.1.1.** Графічний додаток, що створюється.
- 1.2.** **До вхідної інформації** належать вимоги замовника щодо додатку.

2. ПРИЗНАЧЕННЯ ТОВАРІВ

- 2.1. Призначення:** розважальна гра.
- 2.2. Основні критерії ефективності**
 - 2.2.1.** Зручний інтерфейс.
 - 2.2.1.1.** Гравець повинен мати можливість керувати героєм на смартфоні;
 - 2.2.1.2.** Керування повинно бути інтуїтивним.

3. ОСНОВНІ ВИМОГИ ДО ПРОГРАМНОГО КОМПЛЕКСУ

3.1. Загальні вимоги

- 3.1.1.** Графічний додаток працює на смартфонах на базі ОС Android версії 4.4 або вище;

3.1.2. Вимоги до апаратного забезпечення смартфона - не передбачені і можуть встановлюватися розробником програмного комплексу;

3.1.3. Графічна гра повина мати зручний інтерфейс.

3.2. Додаткові вимоги

3.2.1. Мова програмування Java, фреймворк LibGDX.

3.2.2. Вимоги до ліцензійного ПЗ не передбачаються і вирішуються замовником

3.3. Вимоги до складу і архітектури

3.3.1. Розробник самостійно вибирає склад і виконує розробку архітектури ПР

3.3.2. Особливих умов до складу та архітектури ПР не передбачено

3.4. Вимоги до якості і надійності

3.4.1. Графічний додаток повинен надійно працювати.

3.4.2. Розробник обирає технічні характеристики персонального комп'ютера, смартфона, налаштовує системне програмне забезпечення.

3.4.3. Розробник гарантує роботу графічного додатку без збоїв та переналаштувань.

3.5. Вимоги до експлуатації

3.5.1. Розробник використовує смартфон, на якому графічний додаток повинен надійно працювати.

4. ТЕХНІКО - ЕКОНОМІЧНІ ВИМОГ ДО КІНЦЕВОГО ПРОДУКТУ

Вартість робіт по розробці даної ПР визначається згідно з договором на розробку. Вартість пропонованих аналогів повинна забезпечити економічну доцільність їх застосування.

5. ВИМОГИ ДО МАТЕРІАЛІВ І КОМПЛЕКТУЮЧИХ

5.1. Вимоги до екологічної безпеки при експлуатації.

Не пред'являються.

5.2. Спеціальні вимоги до кінцевого продукту.

Не пред'являються.

5.3. Вимоги до безпеки для населення при експлуатації продукції.

Не пред'являються.

6. ЕТАПИ ВИКОНАННЯ ПР

Етапи виконання ПР можуть уточнюватися згідно календарного плану робіт за погодженням між замовником і виконавцем

№	Етапи виконання роботи	Термін виконання і обсяг робіт	Звітні матеріали
1	Аналіз розробки програмного комплексу та розробка першої версії. Аналіз вимог. Розробка структури. Попереднє тестування.		Фрагмент програмного комплексу на ЕОМ замовника, який виконує всі основні функції і звітна документація п.8.2
2	Коригування структури. Розробка допоміжних функцій. Розробка остаточної версії програмного комплексу і його обробки. Тестування.		Готовий програмний комплекс на ЕОМ замовника і звітна документація п.8.2
3	Доопрацювання окремих модулів і навчання користувачів. Розробка звітних матеріалів по п.8 цього ТЗ		Звітні матеріали згідно з пунктом 8.

7. ПРИЙОМ

7.1. Необхідні вимоги для впровадження ПР і завершення робіт.

Оцінка результатів розробки і доцільність її продовження здійснюється замовником за поданням наступних матеріалів:

- встановлено програмний комплекс на ЕОМ замовника;
- перелік файлів на резервному носії;
- короткий опис роботи ПР і опис всіх файлів, які необхідні для роботи

ПР.

- перелік документів
- Технічне завдання
- Пояснювальна записка
- Програма і методика тестування
- Керівництво користувача

7.2. Перелік звітних документів, необхідних для прийняття етапів роботи:

- короткий опис результатів етапу у вигляді анотованого звіту (для 1 та 2 етапів);
- частковий програмний комплекс на ЕОМ замовника згідно календарного плану робіт;
- акт приймання продукції.

Звітні матеріали подаються у вигляді звітів на папері по ГОСТ 7.32-91

7.3. Загальний перелік до прийому звітних документів, макетів, експериментальних зразків.

До прийому пред'являються: акт здачі-приймання продукції, акт впровадження ПР.

7.4.Тестування ПР

Тестування виконується в "Програми і методики тестування", яка розробляється виконавцем і затверджується замовником

8. ПОРЯДОК ВНЕСЕННЯ ЗМІН ДО ТЕХНІЧНЕ ЗАВДАННЯ, ЩО ЗАТВЕРДЖЕНО

Дане технічне завдання може уточнюватися в процесі розробки ПР при узгодженні сторін з оформленням доповнень до ТЗ.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЗ «ЛУГАНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА»

Навчально-науковий інститут математики та інформаційних
технологій

(назва факультету, інституту)

Інформаційних технологій та систем

(назва кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

БАКАЛАВРА

(освітньо-кваліфікаційний рівень)

на тему: **РОЗРОБКА КРОСПЛАТФОРМНОЇ 2D ГРИ ЗАСОБАМИ JAVA**
ТА LIBGDX

Виконав: студент 4 курсу
напряму підготовки (спеціальності)
121 «Інженерія програмного
забезпечення»

(шифр і назва напряму підготовки, спеціальності)

Вєрнік О. В.

(прізвище та ініціали)

Керівник _____ Донченко В. Ю.

(прізвище та ініціали)

Рецензент _____ Козуб Ю. Г.

(прізвище та ініціали)

Полтава – 2024 року

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	3
ВСТУП	4
РОЗДІЛ 1. МЕТОДОЛОГІЯ РОЗРОБКИ ГРАФІЧНИХ ІГРОВИХ ДОДАТКІВ	6
1.1. Теоретичні основи графічних ігрових додатків	6
1.2. Принципи розробки гри	14
1.3. Концепція кросплатформного підходу	26
1.4. Можливості фреймворку LibGDX для створення кросплатформних ігрових програм	27
Висновки до розділу	28
РОЗДІЛ 2. РОЗРОБКА КРОСПЛАТФОРМНОЇ ГРИ SIEGE ЗА ДОПОМОГОЮ ФРЕЙМВОРКУ LIBGDX.....	30
2.1. Підготовчий етап розробки програмного додатку	30
2.2. Архітектура проєкту (огляд класів та об'єктів)	31
2.3. Розробка інтерфейсу та «сцени» гри	32
2.4. Розробка об'єктів персонажів гри	34
2.5. Додавання створених персонажів до «сцени» та робота над логікою гри	36
2.6. Реалізація кросплатформності	37
2.7. Керівництво користувача	39
Висновки до розділу	41
ВИСНОВКИ.....	43
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	45
Додаток А. Ієрархія класів кросплатформної гри Siege	48
Додаток Б. Лістинг абстрактного класу GameCamera.java	49
Додаток В. Лістинг стартового екрану (меню) MainMenuScreen.java	50
Додаток Д. Лістинг частини коду «сцени» гри StageScreen.java	56
Додаток Е. Лістинг коду головного героя Player.java	76

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПК	-	персональний комп'ютер
ОС	-	операційна система
ПЗ	-	програмне забезпечення
ШІ	-	штучний інтелект
ООП	-	об'єктно-орієнтоване програмування

ВСТУП

Актуальність роботи. З кожним роком популярність комп'ютерних ігор зростає все більше і більше. Наймовірно популярні зараз, комп'ютерні ігри в своєму розвитку пройшли довгий шлях від примітивних аркад до повноцінних віртуальних світів. Сучасна комп'ютерна гра – це багатофункціональна програма, яку використовують не тільки з розважальними, а і з навчальними та пропагандистськими цілями. Комп'ютерні ігри присутні в громадській та культурній сферах - мистецтві, освіті, етики, психології, соціальній комунікації тощо.

Популярність комп'ютерних ігор обумовила їх широке застосування не тільки на настільних ПК, а й на смартфонах та інших мобільних пристроях, які в більшості своєї працюють на базі ОС Android. Кросплатформеність поширюється практично на кожен ІТ-структуру, починаючи від кросплатформних мов програмування, закінчуючи кросплатформним середовищем виконання. Тому постає питання розробки кросплатформних ігрових додатків, які б працювали на різних платформах.

Вищесказане доводить актуальність обраної теми дипломної роботи.

Об'єкт дослідження – кросплатформна графічна гра.

Предмет дослідження – технологія створення кросплатформних графічних ігор засобами Java та LibGDX.

Метою роботи є розробка кросплатформної графічної 2D гри засобами Java та LibGDX.

Відповідно до предмета, мети було визначено основні **завдання дослідження:**

- дослідження теоретичних основ графічних ігрових додатків;
- розгляд принципів розробки комп'ютерної гри;
- аналіз концепції кросплатформного підходу до розробки додатків;
- вивчення можливостей фреймворку libGDX для створення ігрових програм;

- розробка кросплатформної 2D гри за допомогою фреймворку LibGDX.

Для вирішення завдань дослідження використано такі **методи дослідження**: *теоретичні*: аналіз наукової літератури, узагальнення та систематизація теоретичних положень про створення кросплатформних додатків, комп'ютерних ігор; *емпіричні*: порівняний аналіз можливостей засобів розробки крос платформних додатків; *експериментальні*: тестування розробленої системи.

До складу роботи входять два розділи, які висвітлюють питання методології розробки графічних ігрових додатків та розробки кросплатформної 2D гри Siege за допомогою фреймворку LibGDX.

Практична значення розробки – розроблений проєкт може застосовуватися у навчальних цілях під час підготовки відповідних напрямків спеціалістів.

РОЗДІЛ 1.

МЕТОДОЛОГІЯ РОЗРОБКИ ГРАФІЧНИХ ІГРОВИХ ДОДАТКІВ

1.1. Теоретичні основи графічних ігрових додатків

Зараз існує велика кількість різноманітних комп'ютерних ігор, і їх творці постійно випускають нові. Оскільки створення комп'ютерних ігор пов'язано з розважальною сферою, їх класифікація не така проста, як може здатися на перший погляд. Жанри ігор формувалися без певної структури та на основі інтуїтивного підходу протягом тривалого часу. Розробники ігор проводили сміливі експерименти, досліджуючи нові ігрові механіки. Невдалим експериментам не приділялося багато уваги, а успішні ігри ставали прикладом для інших розробників. Розробники копіювали популярну ігрову механіку, додавали свої ідеї, і таким чином навколо найпопулярніших ігор утворювалися цілі класи схожих ігор, які отримали назву жанрів.

Поділ на жанри є дуже корисним на практиці. Ігри певного жанру вже мають свою зацікавлену аудиторію. Коли розробник оголошує, що випускає гру в певному жанрі, гравці вже мають уявлення про те, що відбуватиметься в грі, навіть без додаткових пояснень від розробників. Часто розробник компанія не може однозначно визначити жанр нової гри, оскільки бажає продати цей продукт якомога більшій кількості користувачів, і тому не хоче обмежувати потенційну аудиторію тематичними рамками.

Зазвичай ігри класифікуються за жанрами і кількістю гравців. Вони поділяються на кілька типів, такі як квести, екшн, рольові ігри (РПГ), файтинги, стратегії, симулятори, логічні та азартні ігри, а також інші. Ця класифікація допомагає гравцям швидше зорієнтуватися в світі ігор. Наприклад, якщо гра відноситься до жанру квесту, гравці можуть очікувати розв'язання головоломок і пошук предметів для продовження сюжету. З іншого боку, якщо гра відноситься до жанру екшну, гравцям може бути цікавим швидкий темп гри та активні битви.

Класифікація також допомагає розробникам ігор визначити свою цільову аудиторію та залучити більше гравців, пропонуючи їм ігровий досвід, який вони вже знають і люблять у певному жанрі.

Отже, поділ ігор на жанри є важливим і корисним інструментом для розуміння та класифікації комп'ютерних ігор. Цей підхід допомагає гравцям знайти ігри, які відповідають їхнім вподобанням, а розробникам — залучити більше гравців шляхом пропозиції ігор у вже визначених жанрах.

Більше того, класифікація ігор за жанрами дозволяє створити зручну систему для сприйняття та обговорення ігрового контенту. Гравці можуть легко спілкуватися та обмінюватися враженнями про ігри, використовуючи загальноприйняті терміни і жанрові визначення. Наприклад, якщо хтось каже про "рогалик", інші гравці відразу розуміють, що йдеться про рольову гру зі случайно генерованими рівнями та постійним розвитком персонажа. Такі загальні терміни та жанрові уявлення сприяють обміну інформацією, порівнянню ігор та вибору нових ігор для гри.

Звісно, варто зазначити, що ігрові жанри не є жорсткими і непорушними категоріями. Багато сучасних ігор поєднують елементи з різних жанрів і використовують гібридні механіки. Це дозволяє розробникам експериментувати та пропонувати нові інновації в ігровій індустрії. Такі гібридні ігри можуть викликати нові жанрові підходи та розширити можливості для створення унікального ігрового досвіду.

Загалом, класифікація ігор за жанрами є важливою складовою розвитку ігрової індустрії. Вона допомагає гравцям зорієнтуватися в широкому виборі ігор та знаходити ті, які відповідають їхнім вподобанням. Розробники ж можуть використовувати жанрову класифікацію для просування своїх ігор та привертання цільової аудиторії.

У більшості випадків ігри класифікуються за жанрами та кількістю гравців. Жанрова класифікація включає квести, екшн, рольові ігри (РПГ), файтинги, стратегії, симулятори, логічні, азартні та інші (зображені на рисунку 1.1).

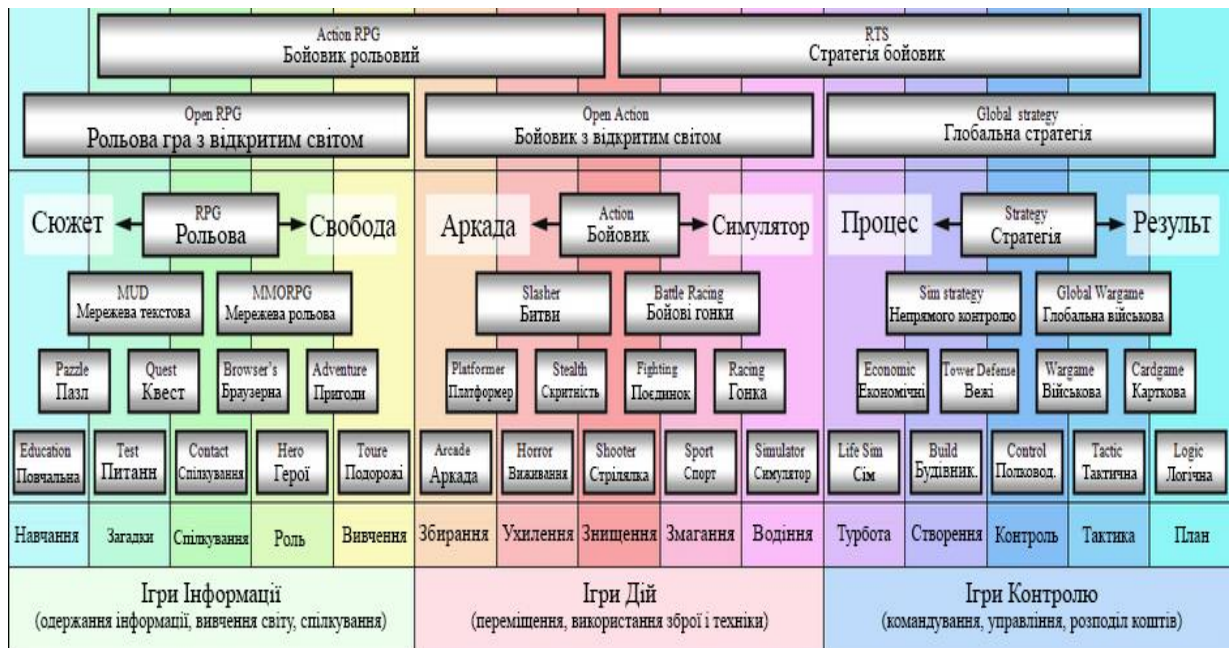


Рис. 1.1. Типи ігор

Розглянемо найпопулярніші жанри ігор, які на сьогоднішній день є широко поширеними:

1. Квести - це ігри, в яких один або кілька персонажів подорожують з метою досягнення поставленої мети, подолання різних труднощів і розв'язання головоломок.

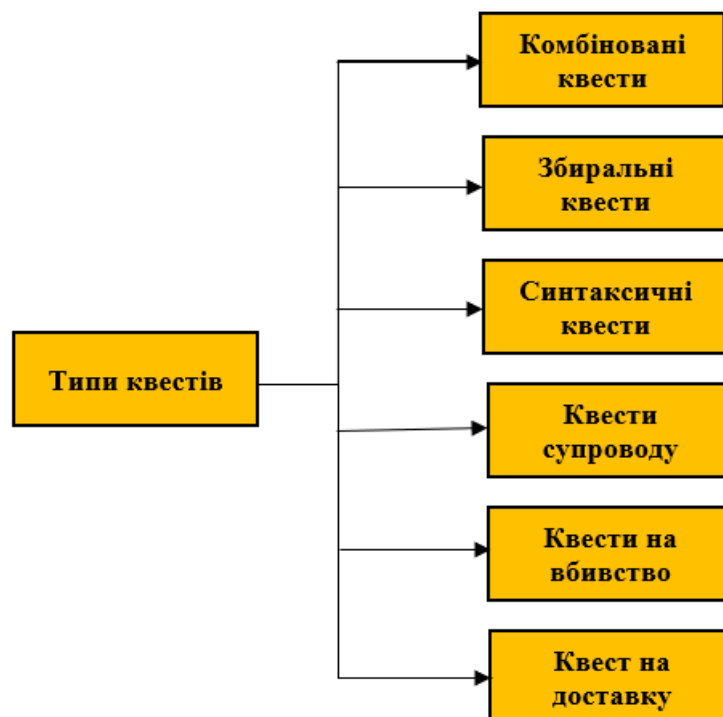


Рис. 1.2. Піджанри квести

2. Екшн (action) - це популярний жанр, що включає бродилки-стрілялки від першої особи, в яких головний акцент зроблений на швидкість, реакцію та бойові вміння.

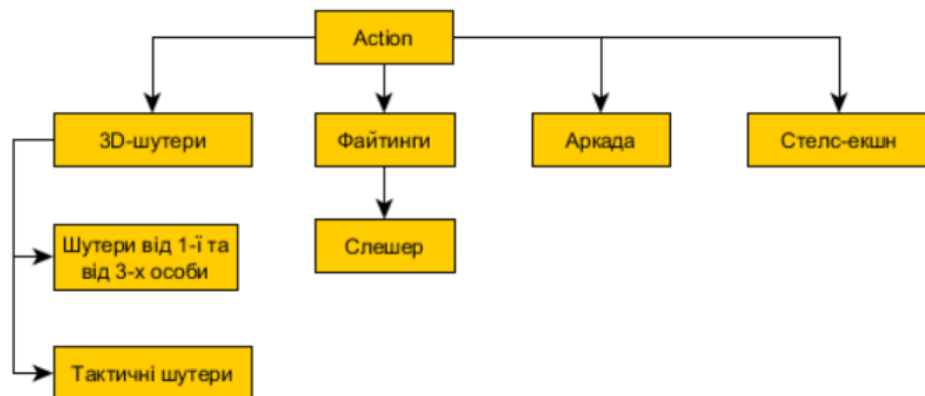


Рис 1.3. Піджанри Action

3. Рольові ігри (РПГ) - в цих іграх гравець приймає роль певного персонажа і виконує поставлені перед ним завдання, розвиваючи його навички, збираючи екіпірування та взаємодіючи з іншими персонажами.

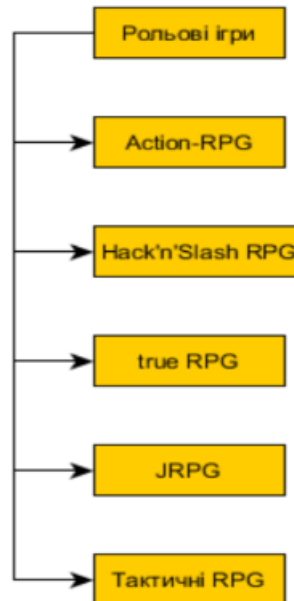


Рис. 1.4. Піджанри рольових ігор

4. Стратегії та логічні ігри - ці ігри наслідують діяльність управління, де гравець приймає рішення, розв'язує головоломки та планує свої дії з метою досягнення успіху.

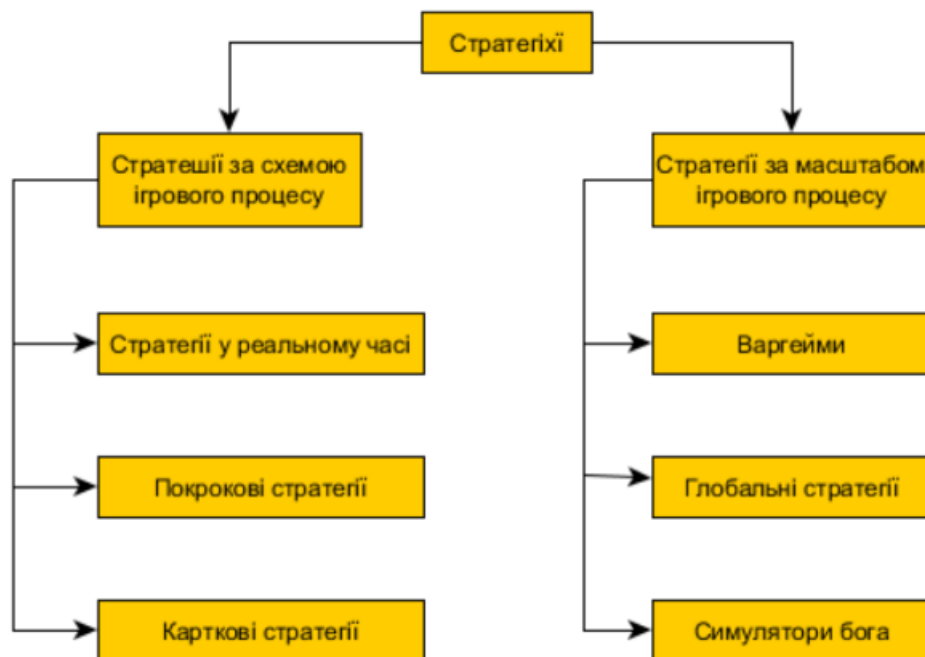


Рис. 1.5. Піджанри стратегії

5. Симулятори - це ігри, що імітують керування автомобілем, космічним кораблем, літаком та іншими об'єктами. Вони спрямовані на реалістичне відтворення певного виду діяльності.



Рис. 1.6. Піджанри Симулятор/Менеджери

6. Азартні і логічні ігри - ці ігри сприяють розвитку розумових навичок і вимагають від гравця логічного мислення та стратегічного планування.

Існує багато платних і безкоштовних ігор, де зустрічаються різні жанри, включаючи екшн, РПГ, деякі види стратегій, квести та інші. У таких іграх

присутні різноманітні персонажі та геймплейні елементи, які залежать від конкретної гри. Це дозволяє гравцям знайти ігру, яка найкраще відповідає їхнім інтересам і вподобанням.

Таким чином, незалежно від того, чи це платна чи безкоштовна гра, ігрові жанри пропонують різноманітність варіантів для гравців. Вони можуть обрати екшн-пригоди, в яких досліджують світ та борються з ворогами, або погрузитися у глибокий сюжет РПГ і відчувати себе героєм фантастичного світу. Вибір жанру допомагає гравцям зорієнтуватися в світі ігор та знаходити ігри, які найбільше їх захоплюють.

Ігри без персонажа, такі як логічні і симулятори, відрізняються відсутністю сюжету та не викликають сильної емоційної залежності або впливу на психіку гравця. Ці типи ігор часто пропонують інтелектуальні завдання, що сприяють розвитку мислення.

У стратегічних іграх також відсутні персонажі, а гравець управляє грою зі свого обличчя. Однак, особливістю стратегій є система внутрішніх факторів, що регулюють геймплей. Гравець приймає рішення та спостерігає за їх наслідками, побудовуючи логіку подальших дій.

Таким чином, ігри без персонажа, такі як логічні і симулятори, пропонують інтелектуальні завдання, сприяючи розвитку мислення, тоді як стратегічні ігри з відсутністю персонажа мають особливу систему внутрішньоігрових факторів, де гравець приймає рішення та будує логіку своїх дій.

Рольові ігри (РПГ), а також окремі екшн і квести, є найвідомішими іграми, де гравцеві надається свобода вибору. Гравець самостійно обирає персонажа, одягає його та наділяє різними здібностями. Головна особливість цих ігор полягає в тому, що гравець має велику свободу дій, які впливають на хід та результат гри. Грати в такий тип ігор може викликати стійке захоплення, оскільки гравець співпереживає своєму персонажу, розвиває його відповідно до своїх уподобань і стає активним учасником ігрового світу. Ігровий процес сприймається як реальність, а багатьом РПГ-іграм подібність з фантастичними книгами з можливістю керування головним героєм становить особливий інтерес для любителів читання. Захоплення такими іграми можна порівняти з

пристрастю до книг, воно не має шкідливих наслідків, а навпаки, сприяє розвитку фантазії, захоплює пізнанням історій та відображенням реальності.

Ігри з лінійним сюжетом, такі як стрілялки, не передбачають вибору гравця. Гравець ідентифікується з готовим персонажем, який вже має визначені характеристики. Розвиток сюжету в таких іграх залежить не від вибору гравця, а від успіху або випадковості. Деякі стрілялки дозволяють грати командою. Такі ігри надають емоційне розслаблення, а успішне проходження часто залежить від швидкої реакції гравця.

На перетині рольових ігор і стратегій знаходиться поняття "абстракція". Це протилежність "бойовикам", які борються зі скукою за допомогою спектакулярності. Ігри, що належать до цієї категорії, мають схематичний характер, обмежену свободу, відсутність реалістичності та візуального шоу.

У стратегічних абстракціях головним завданням є досягнення "цілі", тоді як у рольових абстракціях важливим є "підпорядкування". В результаті отримуємо ситуацію, де гравець підкоряється чужим цілям і виконує нудну роботу. При цьому можуть відбуватися події, які раніше не траплялися, але все одно гравець змушений підкорятися волі інших (це основна відмінність від "буденності"). Центральним елементом є поняття рабства і безумовне виконання наказів. Такі аспекти не мають нічого спільного з грою.

Склад відеоігрової палітри "Буденність" передбачає поєднання елементів стратегій і бойовиків. На перетині цих жанрів знаходиться поняття "симуляція". Це повна протилежність "рольовим іграм", які протистоять "буденності" за допомогою цікавого "сюжету".

Ігри, що належать до цієї категорії, характеризуються імітацією реальних процесів, обмеженою свободою, відсутністю новизни і повністю відсутнім сюжетом.

З погляду стратегій у симуляціях найбільш важливим є сам "процес". З боку бойовиків наголошується на "реалістичності", яка відтворюється на екрані. В результаті отримуємо реалістичну імітацію процесу, яка відображає звичайне життя. В грі існує кілька повторюваних подій, але нічого нового не з'являється. Центральний елемент не використовується, оскільки повна реалістичність не

досяжна у відеоіграх, а також багато хто не шукає точну копію реальності в грі, яка вже існує у реальному житті.

На перетині рольових ігор і бойовиків розташовується концепт "свобода". Це протилежність стратегіям, які, в свою чергу, наголошують на "самотності" та "керівництві" над численними підлеглими. (Варто зауважити, що стратегії не борються з самотністю в звичайному розумінні, але вони відтворюють владу як найпоширенішу форму спілкування).

Для ігор, які знаходяться на цьому перетині, характерні великі відкриті світи, відсутність повної симуляції реальності, відсутність новизни та відсутність контролю над іншими учасниками.

З боку рольових ігор в "іграх свободи" основним є відсутність готових рольових зв'язків та можливість самотійного вибору, тобто "рольова свобода".

З боку бойовиків спостерігається "спрощення" реальних подій та процесів у порівнянні з симуляторами.

Узагальнюючи, результатом є рольова свобода в спрощеній моделі реального світу. Свобода однієї особи завершується там, де починається свобода іншої, тому максимальна точка свободи - це повна самотність. Найбільш спрощена точка виявляється у порожньому світі, де є свобода простору. Найбільш вільною дією є створення чогось абсолютно нового. Загалом, центральною точкою цього елемента є повна самотність у порожньому світі, де можна творити - редактор рівнів. Прямо під час гри можливості редагування не використовуються, але окремі елементи процесу створення присутні в іграх.

За допомогою цього шестикутника легко розуміти найважливіші аспекти певних видів ігор та компроміси, з якими розробники ігор стикаються. Він також показує, що є недопустимим у іграх і що можна комбінувати між собою, а що не піддається змішанню.

Для згладжування переходів між гранями, розробники повинні тщательно прорахувати геймплей - сам процес гри з погляду гравця, який є основою зацікавленості та привабливості гри. Геймплей включає різні аспекти, включаючи технічні аспекти, такі як внутрішньоігрова механіка, методи взаємодії між "штучним інтелектом гри" та гравцем тощо. Однак, сам термін

"геймплей" є загальним та зазвичай використовується для опису відчуттів, отриманих під час гри, під впливом таких факторів, як графіка, звук та сюжет. Тому пріоритетом є прорахування моделі поведінки, що є фундаментальним для жанру гри, а потім вдосконалювати та поліпшувати взаємодію з гравцем.

1.2. Принципи розробки гри

Розробкою відеогри може займатися як одна людина, так і колектив розробників (фірма). Комерційні ігри часто створюються командами розробників, які були найняті однією фірмою. Фірми можуть спеціалізуватися на виробництві ігор для персональних комп'ютерів, ігрових консолей або планшетних комп'ютерів. Нерідко розробка фінансується іншою, більш крупною фірмою – видавництвом. Фірма-видавництво по закінченню розробки займається розповсюдженням копій гри та бере на себе пов'язані з цим витрати. Буває, що фірми-видавництва наймають команди розробників, або фірма-розробник самостійно (без залучення видавництва) розповсюджує копії ігор, наприклад, засобами цифрової дистрибуції (інді-ігри).

У склад типової сучасної команди розробників зазвичай входять представники різноманітних спеціалізацій:

- a) Один або декілька продюсерів для нагляду за виробництвом;
- b) Геймдизайнери;
- c) Художники;
- d) Сценаристи;
- e) Звукооператори:
 - Композитори;
 - Творці звукових ефектів;
- f) Тестувальники.

Розробка гри складається з таких етапів як проєктування, творчість та видання. Проєктування зображено на рис. 1.7.

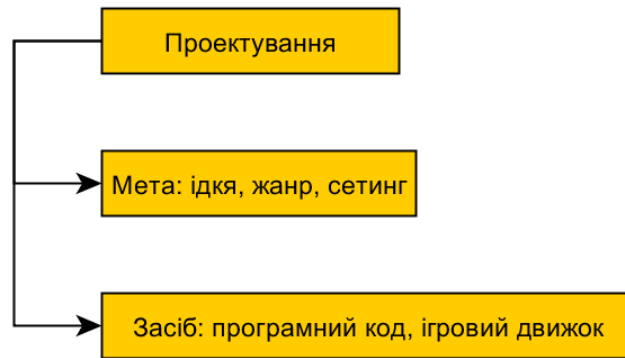


Рис. 1.7. Проектування

Визначення мети

Спочатку треба розібратися з метою. Що ми хочемо отримати? Етапом концепції і визначення мети займається керівник проєкту.

Жанр. Можна з самого початку представляти в найдрібніших деталях готову гру, а можна по ходу розробки домірковувати і сюжет, і стиль, і особливості гри. У цій справі не обов'язкова зайва точність, але, як мінімум, потрібно задати напрямок розвитку ігрового проєкту. Жанр гри необхідно вибрати на самому початку в обов'язковому порядку. Жанр і буде основним напрямком розвитку гри.

Обраний жанр можна трохи коригувати по ходу роботи, але його сутність повинна залишатися незмінною. Жанр - це своєрідний фундамент всієї гри.

Сеттинг. Поділ комп'ютерних ігор на жанри дуже специфічне і не схоже на систему жанрів фільмів і книг. Ігрові жанри визначають лише основні дії, які будуть здійснювати гравці в процесі гри, тим самим вони відповідають тільки на питання "ЩО?". На питання "ДЕ?" і коли?" відповідає інша основна характеристика гри - сеттинг.

Сеттинг – це приналежність гри до якоїсь сюжетної темі або до певного віртуального світу. У середовищі комп'ютерних ігор сформувалося кілька найбільш популярних сеттингів: фентезі, наукова фантастика (sci-fi), друга світова війна, середньовіччя, стімпанк, постядерного світ, аніме, комікси.

Створення гри в популярному сеттинге забезпечує її власну популярність, та й гравці відчують себе затишно і комфортно у вже знайомому світі. Деякі ігри створюються в своїх унікальних сеттингах або в незвичайних поєднаннях стандартних тем. Такі ігри менш популярні, але, тим не менш, вони мають свою

аудиторію особливих гравців, які терпіти не можуть шаблонність і одноманітність.

Визначення засобу

Мета ігрового проєкту задана, тепер нам потрібно вибрати засоби (матеріали і інструменти) для її досягнення. І тут ми стикаємося з незвичайним феноменом комп'ютерного світу - і матеріалом, і інструментом ігрового проєкту є одна і та ж сутність - програмний код. Код як будівельний матеріал - це цифрові зображення, тривимірні моделі, звуки і тексти у вигляді послідовностей одиниць і нулів. Код як інструмент - це команди в рядках програмного коду, що управляють ігровими об'єктами всіх перерахованих типів.

Програмний код. Програмний код у ролі інструменту являє собою каркас (скелет), на який будуть нанизуватися результати всіх наступних етапів розробки. Цим етапом займаються програмісти.

Перш за все ми повинні вибрати мову програмування, який нам найбільше підходить. Після цього має бути важка і копітка робота по написанню програмного коду, здатного оперувати двомірними або тривимірними об'єктами в просторі, прив'язкою зображень і звуків. По ходу розробки доведеться вивчити всі формати зображень і аудіофайлів, всілякі кодеки і кодування.

Ігровий движок. В наш час можна відразу скористатися готовим програмним модулем (ігровим движком), де вже реалізовані базові функції, здатні зв'язати воедино графіку, звук, об'єкти і їх руху. Таким чином вибір мови програмування замінюється іншою дилемою - вибором готового ігрового движка.

Застосування ігрових движків ще не звільняє повністю від використання послуг програмістів, але зводить їх до мінімуму. Стандартний програмний модуль ще доведеться налаштувати, додати в нього щось своє, щоб ігровий проєкт вийшов більш унікальним.

Принцип творчого підходу до створення ігор зображено на рис. 1.8.

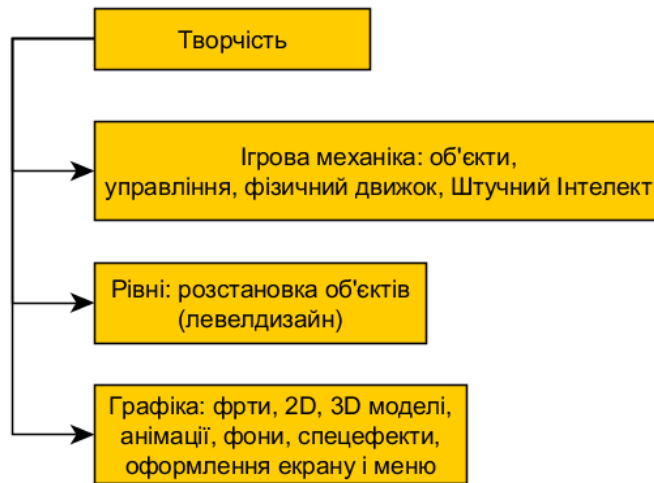


Рис. 1.8. Творчий підхід

Визначення ігрової механіки

Найважливіша творча частина будь-якої гри - ігрова механіка. Ця річ знаходиться не на поверхні, тому часто вислизає від погляду неуважних цінителів ігор.

Ігрова механіка, по суті своїй, це звід правил, за якими функціонуватиме гра.

Об'єкти. Основою всієї механіки є ігрові об'єкти. Головний герой гри, комп'ютерні суперники, другорядні персонажі (NPC), бонуси, рухомі об'єкти, декорації - все це ігрові об'єкти зі своїми властивостями і можливими діями.

Управління. Ігрова механіка визначає якими клавішами буде управлятися головний герой або основний ігровий об'єкт, який вплив буде відбуватися після натискання тієї чи іншої кнопки. Сюди ж відноситься закони поведінки ігрових об'єктів (фізичний движок) і поведінку ворогів (штучний інтелект).

Фізичний движок. Якщо «управління» відповідає за переміщення підконтрольного нам персонажа, то фізичний движок відповідає за ті руху, які відбуваються без прямого втручання гравця. Ці дії імітують фізичні закони реального світу (іноді трохи спотворені в сторону фантастики).

У готових ігрових движках найчастіше реалізовані і фізичні движки. Нам лише залишиться привласнити своїм унікальним об'єктам вже готові фізичні характеристики: вага, щільність, еластичність, знищуванність. Якщо ж треба створити свій фізичний движок, то для цього нам знадобиться талановитий

програміст, який добре розуміє принципи об'єктно-орієнтованого програмування (ООП) і трохи розбирається в класичній фізиці.

Штучний інтелект (ШІ). ШІ відповідає за поведінку комп'ютерних ворогів або союзників.

Роль ШІ значно різниться в залежності від жанру гри. У екшенах дії ворогів вкрай примітивні; в RTS стратегіях досить пари десятків скриптів, щоб надати супернику гадану розумність; в стелс-екшенах, слешерів і файтинга необхідно створити унікальну систему поведінки для кожного типу ворогів, інакше дурні вороги зроблять гру нецікавою. Серйозна стратегічна гра вимагає колосальної роботи над ШІ, а в простих казуальних іграх і в онлайн-проектах, орієнтованих на битви тільки між реальними гравцями, штучний інтелект взагалі не потрібен.

Визначення рівнів

Створені ігрові об'єкти розставляються в окремих віртуальних просторах - рівнях (локаціях). Ігри найчастіше містять безліч окремих рівнів, перехід між якими відбувається по ходу сюжету.

Побудовою рівнів займаються левелдизайнери.

В ідеалі левелдизайнери беруться з числа затятих гравців. Це відбувається тому, що будь-яка людина з боку, нехай навіть і творчий, але дуже далекий від теми ігор, не зможе добре впоратися з цим завданням. Левелдизайнер повинен добре уявляти собі ігровий процес, і відчувати, як від переміщення об'єктів на рівні змінюватиметься ігрова ситуація.

Редактор рівнів. Досить часто в комплекті з грою поставляється редактор рівнів, за допомогою якого звичайні гравці можуть самостійно створювати собі нові карти і рівні. Розробники ігор підтримують поширення саморобних карт між гравцями і часто викладають кращі роботи на своїх офіційних серверах. Редактори рівнів створюються не тільки для розваги гравців і збільшення терміну життя окремо взятої гри, але і для того, щоб відшукати серед ігрової аудиторії найбільш талановитих людей. Таким чином, ігрові студії вирішують свою кадрову проблему. Як вже говорилося вище: кращий левелдизайнер - це завзятий гравець.

Оформлення гри

Створенням графіки займаються художники, Геймдизайнер.

Арти. Для початку потрібно створити образи героїв, ворогів, ігрових предметів, задніх фонів. Спочатку вони малюються або на папері, або на комп'ютері з використанням графічного планшета. Для невеликих ігрових студій цей етап не обов'язковий, але він просто необхідний у великих компаніях, щоб не на пальцях, а на наочних зображеннях пояснити всім дизайнерам, що у них в результаті повинно вийти.

2D, 3D моделі. На основі артов дизайнери створюють або двомірні спрайт з пікселів, або тривимірні моделі з полігонів.

Анімація. Для ігрових об'єктів, які будуть пересуватися в ході гри, створюються анімації. Особливо складно доведеться з героями і ворогами, кількість анімацій яких іноді перевищує цілу сотню різних рухів.

В наш час для створення людиноподібних 3D-персонажів існує спеціальна технологія «Motion Capture», що дозволяє створювати анімації на основі рухів справжніх людей. Ця технологія доступна лише великим і дуже багатим компаніям. Для використання Motion Capture потрібно не тільки придбати дороге устаткування, але ще й найняти групу акторів, з яких будуть записуватися руху.

Спецефекти. Візуальні спецефекти - це, по суті своїй, ті ж анімації, тільки замість переміщення об'єктів в них використовуються переміщення частинок і світлофільтрів. Промені світла в різні боки при взятті бонусів, вогонь на палаючому будинку, димова завіса після вибуху гранати, лазерні промені з дула гвинтівок, накладення фільтрів розмиття при знаходженні під водою і фільтрів затемнення в погано освітлених місцях - все це спецефекти. Без подібних ефектів гра буде здаватися прісною і занадто буденною. Використання спецефектів додає грі яскравості, соковитості і експресивності.

Оформлення екрану та меню. Оформити потрібно не тільки ігрові рівні, а й систему, що об'єднує їх в єдине ціле - ігрове меню (рядки, кнопки, сторінки налаштувань). Початкове меню - це взагалі візитна картка гри, і виглядати вона повинна ідеально. На ігровому екрані так само є безліч елементів, до яких можна застосувати оформлення - кількість життів, лайфбар, мінікарта, меню швидкого вибору дій, інвентар героя, списки завдань, екрани діалогів. Англійською мовою

все це називають одним скороченням - GUI (Graphical User Interface - графічний користувальницький інтерфейс).

Оформленням інтерфейсу і меню займаються художники, програмісти і верстальники html-сторінок.

Сюжет гри

Залучити гравця до свого проєкту дуже складно, але ще складніше зробити так, щоб гравець пройшов гру до кінця. Будь-яке розчарування, занудне або важке місце може миттєво відштовхнути гравця від подальшої гри. У більшості випадків гравець залишить і забуде гру без жодного жалю. І тільки грамотно поданий якісний сюжет може змусити гравця зібратися з силами, пройти всю гру, а значить - дослухати вашу інтерактивну історію до кінця.

Скрипти, події. Найкращий варіант - коли сюжет існує прямо всередині гри. Це досягається за допомогою використання скриптових сценек.

Скрипт являє собою наступне: гравець заходить в певне місце, або робить потрібну дію, або виконуються ще якісь необхідні умови, і після цього починають виконуватися дії, запрограмовані вами на цей випадок. Наприклад, у військовому 3D шутере піднімаємося на піднесеність, підходимо до встановленого кулемета (умова виконана), через 10-15 секунд після цього внизу раптом починається ворожа масована атака, і нам є на кого використовувати кулемет (відбулися події).

За допомогою скриптових подій можна вносити різноманітність в ігровий процес або навіть перетворити гру в атракціон нескінченних скриптів (саме на цьому побудована серія ігор Call of Duty). Єдиний мінус такого способу - у гравця зменшується свобода дій. Все відбувається з волі скриптів, і мало залежить від дії гравця.

Продумуванням скриптів займаються сценаристи, а їх реалізацією - програмісти.

Діалоги, оповідання. У старих класичних іграх сюжет існує відокремлено від ігрового процесу. Наприклад, при завантаженні або закінченні рівнів нас знайомлять із сюжетною історією, розповідають про відносини між героями і ворогами, пояснюють, що і для чого потрібно зробити на рівні. У процесі самої

гри нічого з вищесказаного не має ніякого значення, і гравець може сміливо пропускати всі ці тексти. Найчастіше так і відбувається - тексти залишаються прочитаними. А все через те, що немає ніякої вагової причини їх читати.

Інша справа - внутрішньоігрові тексти або діалоги. Вони відбуваються в процесі гри, але в безпечних місцях, або з зупинкою ігрового часу, щоб гравець міг зосередитися тільки на тексті. Оповідання гравцеві доводиться вислуховувати, так як гра на цей час завмирає, але не зупиняється зовсім. А в діалогах ще й потрібно вибирати варіант відповіді. Вибір варіанту надає прослуховування тексту інтерактивність і хоч якийсь практичний сенс - правильно обраний відповідь може принести додатковий бонус, полегшити подальшу гру або зберегти обраний характер героя в рольових іграх.

При створенні гри тексти оповідань і діалогів краще зберігати в окремих файлах, підвантажуваних по ходу гри. Відділення художнього тексту від технічних кодів допоможе в майбутньому, якщо треба буде створити локалізовану версію гри на інших мовах світу.

Написанням текстів і діалогів займаються сценаристи і письменники.

Відеовставки. Між рівнями гри або в певних контрольних точках рівнів можна замість сухого тексту і озвучення показувати гравцям відеовставки (катсцени). Такі шпалери можна створювати як за допомогою окремих відеофалов, так і за допомогою ігрового движка.

Відеофайли дозволяють передавати гравцеві картинку будь-якої якості і складності, але при створенні дистрибутива гри вони займають багато дискового простору. Заставки, створювані на движку гри, за своєю якістю вже майже не поступаються відрендерене заздалегідь відеозаписів, але для їх хорошого перегляду гравця повинен бути досить потужний комп'ютер, що не завжди буває насправді.

Створенням відеовставок займаються художники, аніматори, 3D-модельєри, сценаристи, режисери.

Звукова складова гри

Красиво намальований і ефектно оформлений проєкт мовчазно дивиться на нас, і чекає, коли ми додамо йому звучання. Це ще одна важлива складова гри.

Звукові ефекти. Для будь-якого маломальського ігрового руху потрібно додати відповідний звук. Це можуть бути удари меча, нанесення рукопашного удару, звуки руху автомобіля, отримання бонусу, виявлення героя ворогом.

Хороші звукові ефекти не тільки заповнюють тишу, але і являють собою продовження графічного стилю гри. Весела аркада наповнена не менш веселими мелодійними звуками, спортивний автосимулятор наповнений рокітливим гулом моторів і брязкотом гальм, тривимірний бойовик приголомшує кулеметними чергами, падаючими гільзами і клацають затворами гвинтівок.

Найчастіше в якості звукових ефектів використовуються реальні звуки, записані в цифровому вигляді. В Інтернеті є безліч безкоштовних колекцій звукових ефектів, нам потрібно лише знайти їх і вибрати найбільш підходящі.

Музика. Крім звуків для повноцінної гри потрібна і музика (саундтрек). Вона буде звуковим фоном для того, що відбувається на екрані. Музика так само є одним із стилістичних елементів гри, і найсильніше впливає на настрій гравця. Готову музику потрібно довго вибирати по невластивому темпом і настроєм. Існують безліч як платних, так і безкоштовних колекції ігрових фонових композицій, які можна використовувати в своїй грі. Або можна замовити композиторам написати нову музику спеціально під гру.

Третьою звуковий залишає гри є озвучка ігрових діалогів і монологів. Ця складова дуже дорога, але її наявність в грі не обов'язково. У деяких іграх діалогів і текстів майже немає, а там, де є, їх можна залишити неозвученими у вигляді текстових субтитрів. Невеликі гри обходяться зовсім без озвучення, а в великих проєктах для озвучування навіть запрошують професійних акторів світової величини.

Впровадження в експлуатацію зображено на рис. 1.9.



Рис. 1.9. Впровадження в експлуатацію

Доопрацювання

Процес розробки великої гри побудований таким чином, що різними її елементами займаються різні фахівці. На початковому етапі гра представляє собою розрізнений набір творчих напрацювань в різних галузях мистецтва: зображення, звуки, 3D-моделі, архітектура, тексти, сценки, відеовставкі, оформлення. І ось, нарешті, настає такий момент, коли розкидані камені потрібно збирати. За допомогою програмних засобів розрізнені об'єкти з'єднуються в єдину складну систему.

Зведення матеріалу (а-версія). При побудові гри на ігровому движку об'єднання об'єктів відбувається поступово з самого початку процесу. Поки гра не зібрана до кінця, її називають альфа версією. У цей момент вже можна займатися тестуванням окремих рівнів, скриптів і інших механізмів.

На цьому етапі вже технічно можливо випустити демо версію або хоча б відеоролик з ігровим процесом, щоб заздалегідь залучити гравців до свого проєкту.

Усунення помилок (b-версія). Коли гра повністю зібрана, залишається лише усунути отримані помилки (bugs). Вони з'являються в будь-якому випадку, так як гра - це система зі складною структурою. Самі елементи гри наочні і прості, але зв'язку між ними настільки складні і витіюваті, що процес налагодження та усунення помилок може займати до 40% всього часу розробки проєкту. Повністю зібрана, але ще не перевірена на помилки гра називається бета версією.

Пошуком помилок в грі займаються тестери. Дуже часто в якості тестерів залучаються групи звичайних гравців, і це служить початком їхньої кар'єри в ігровій індустрії. Найпростіше ця проблема вирішується в онлайн-іграх - розробники організовують відкриті бета-тести, в яких беруть участь взагалі всі бажаючі гравці.

Продаж

Створенням гри і всіма творчими питаннями займається студія розробників, а всі інші питання (кредити, фінанси, договору, захист прав, рекламні акції, локалізації, продажу) зазвичай перекладаються на плечі іншої організації - ігрового видавця.

Реклама. Перш ніж продати гру кінцевому користувачеві, видавцям для початку потрібно повідомити про існування цієї гри. Звичайно, гру можуть купити, взагалі нічого про неї не знаючи, просто вибравши в магазині навмання, але шанс, що таким чином виберуть саме вашу гру, вкрай низький. Набагато вигідніше поширити інформацію про гру по всіх можливих каналах. Для цього використовують або рекламу в магазині комп'ютерних дисків, або рекламу на Інтернет ресурсах.

Локалізація. В ідеалі, потрібно випускати гру відразу на декількох найпопулярніших в світі мовами (англійською, німецькою, французькою, іспанською, китайською, японською), але для цього потрібно мати цілий штат перекладачів і локалізаторов. Причому, бажано, щоб перекладачі були носіями мови. На свою рідну мову вони зможуть перенести максимум сенсу оригінального тексту. Але простим самотнім розробникам така розкіш не загрожує, та й більша частина великих компаній не поспішають витратити гроші на цю справу.

Система продажу. Класичний спосіб (випуск великого тиражу комп'ютерних дисків, і продаж їх через роздрібні магазини) все ще актуальний, але підходить лише для великих компаній, і для ігор, що мають хоч якусь початкову популярність.

Для невеликих груп розробників ідеально підходить поширення гри через системи цифрової дистрибуції (великі онлайн-магазини). Такий варіант

забезпечує новоспечену маловідому гру вже готової аудиторією покупців, яка сформувалася навколо сервісу. Найвідоміший приклад - сервіс Steam. Завдяки величезній аудиторії гравців, які користуються Steam, майже кожна гра, що вийшла в цьому онлайн магазині, одразу ж набуває світову популярність.

Підтримка

Створення гри і її продаж - це ще не кінець життєвого циклу ігрового проєкту. Коли гра вже знаходиться у кінцевих користувачів, гравцям ще може знадобитися ваша допомога. У великих компаній існують навіть цілі відділи технічної підтримки, що займаються такими питаннями.

Випуск патчів. Попередній бета-тест усунув з гри найочевидніші помилки, але це ще не означає, що їх зовсім не залишилося в грі. Дуже часто буває, що масове використання гри розкриває більш дрібні і непомітні помилки, які не змогли виявити невеликі групи бета-тестерів. Це можуть бути проблеми через несумісність з малопопулярними марками обладнання, або помилки через неприродного використання ігрових можливостей. Фантазія деяких гравців перевершує фантазію розробників, вони можуть зробити в грі такі дії, про які розробники і подумати не могли.

Все це сприяє тому, що часто доводиться вносити виправлення помилок у вже готову гру. Такі виправлення називаються патчами, і цей термін дуже поширений в ігровій індустрії. Мало кому вдається відразу ж випускати ідеальні гри, найчастіше гри доводяться до ідеалу вже після свого офіційного релізу.

Якщо за гру ви отримали від гравців гроші за кожен продану копію, то за випуск патчів ви не отримуєте зовсім нічого. З ринкової точки зору, випуск патчів - збиткове і марне дію, яка не обов'язково виконувати. Але в реаліях ігрової індустрії якщо ви не підтримуєте свій продукт до кінця, то отримуєте погану репутацію у гравців, і втрачаєте можливу майбутню прибуток. Незважаючи на свою безкоштовність, випуск патчів - це дуже корисна і потрібна справа.

Випуск доповнень. Цікава і захоплююча гра без серйозних вад і помилок дає нам зелене світло для подальшої творчої діяльності. Гравці «на ура» взяли наш ігровий продукт? Значить до вже готової гри можна готувати доповнення

або повноцінну другу частину, а розпочатий сюжет можна розвинути далі, перетворити його в цілу епопею або навіть в повноцінну ігрову всесвіт.

Беремо план розробки гри і починаємо створювати новий витвір мистецтва з тих же самих десяти етапів, але тепер вже у всеозброєнні накопиченого досвіду і набутих навичок.

Таким чином, бачимо, що розробка комп'ютерної гри – це складний, багатоетапний процес, в якому задіяна певна категорія спеціалістів, від командної роботи яких залежить майбутня якість та ефективність програмного продукту.

1.3. Концепція кросплатформного підходу

Зараз існує багато ігрових платформ таких як Playstation 1, 2, 3, 4, 4 pro, Xbox360, XboxOne, Nintendo 3DS, PlaystationVita, WiiU та інші. Деякі з відеоігор запускаються на цих різних платформах. Такі комп'ютерні ігри називаються кросплатформними або багатоплатформними.

Багатоплатформність – властивість програмного забезпечення працювати більш ніж на одній програмній (в тому числі – операційній системі) або апаратній платформі, та технології, що дозволяють досягти такої властивості. Кросплатформність дозволяє суттєво скоротити витрати на розробку нового або адаптації існуючого програмного забезпечення.

Багатоплатформна відеогра – відеогра, яку випускають для декількох апаратних платформ. Для кожної окремої платформи така гра має окрему версію, яка призначена для запуску саме на цій платформі. Залежно від особливостей гри й цільових платформ версії однієї гри для різних платформ можуть відрізнятися одна від одної в різних ступенях.

Всі платформи, на яких встановлюються ігри, умовно діляться на:

1. Персональні комп'ютери (ПК). Ігри для ПК називаються ПК-іграми, а версії мультиплатформних ігор для ПК – ПК-версіями ігор. У свою чергу, ПК-ігри діляться в залежності від ОС, на яких вони працюють.
2. Ігрові консолі. Ігри для ігрових консолей називаються консольними іграми, а версії мультиплатформних ігор для консолей – консольними версіями ігор. Всі ігрові консолі діляться на два типи –

портативні ігрові консолі (являють собою автономні пристрої, які містять в одному компактному корпусі всі необхідні для гри пристосування) і «префіксальні» консолі (які підключаються до зовнішніх дисплеїв, звукових динаміків і іншим пристроям). Відповідно, консольні ігри поділяються для цих двох типів ігрових консолей. Ігрові консолі – спеціалізовані комп'ютери, основний акцент при розробці яких був зроблений саме на виконання комп'ютерних ігор.

3. Мобільні телефони (смартфони). На сучасному етапі мобільні телефони виступають у ролі ігрових платформ. Серед ігор для мобільних телефонів напоширеніші казуальні ігри, прості аркади, платформери й ін.

Отже, багатоплатформність – властивість програмного забезпечення працювати більш ніж на одній програмній (в тому числі – ОС) або апаратній платформі, та технології, що дозволяють досягти такої властивості. Принцип кросплатформного підходу дозволяє суттєво зменшити витрати у майбутньому при необхідності перенести гру з однієї платформи на іншу.

1.4. Можливості фреймворку LibGDX для створення кросплатформних ігрових програм

Щоб домогтися багатоплатформності для відеогри є багато різних засобів, одним з яких є використання спеціальних програм, або фреймворків. LibGd - це Java фреймворк, що надає багатоплатформний API для розробки ігор і додатків реального часу з відкритим кодом, заснований на мові програмування Java з деякими компонентами, написаними на C і C++ для підвищення продуктивності певного коду. В даний час підтримує Windows, Linux, MacOSX, Android, iOS і HTML5 як цільові платформи.

Список можливостей:

1. Крос-платформна розробка для Windows, Linux, MacOSX, Android, браузері з підтримкою WebGL та iOS. Підтримуються як 32 так і 64-розрядні версії;

2. Комбіновані бекенд можливості, засновані на JoGL, LWJGL, RoboVM і AndroidAPI;
3. Набір вбудованих класів для реалізації графічного інтерфейсу користувача. Сюди входять як примітивні елементи – такі як кнопки, текстові поля – так і складні елементи – випадаючі списки, панелі з можливістю прокрутки та інше.

LibGDX дозволяє написати код одного разу і потім розгортати гру або додаток на декількох платформах без модифікації. libGDX дозволяє працювати з файловою системою, пристроями введення, аудіо пристроями та OpenGL через єдиний OpenGL ES 2.0 інтерфейс. OpenGL (OpenGraphicsLibrary) – специфікація, що визначає платформонезалежність програмний інтерфейс для створення програмного забезпечення, що використовують двовимірну і тривимірну комп'ютерну графіку.

LibGDX – саме те, що потрібно для мого проєкту, бо він відповідає моїм потребам. Завдяки йому можна створити крос платформний проєкт. Сам фреймворк також є багатоплатформним. А головне він безкоштовний.

Висновки до розділу

У першому розділі було розглянуто методологічні підходи до розробки графічних ігрових додатків. Під час дослідження встановлено, що відеогра – програма, що служить для організації ігрового процесу, зв'язку з партнерами по грі, або сама може виступати як партнер.

Відеоігри можуть поділитися за жанрами Action, Симулятор/Менеджмент, Стратегії, Пригоди, Музичні ігри, Рольові ігри (RPG), Головоломки/ логічні/ пазли, Традиційні настільні ігри, Текстові ігри.

За кількістю гравців ігри поділяються на одно та багато користувальні, багато користувальні на одному комп'ютері, багато користувальні офлайн-ігри, масові онлайн ігри.

За Стилiстикою ігри поділяються на вестерн, кіберпанк, космічні, Постапокаліптичні, стiмпанк, сучасні, фентезі.

За платформою ігри поділяються на персональні комп'ютери, ігрові консолі / приставки, мобільні телефони і КПК.

У склад типової сучасної команди розробників зазвичай входять представники різноманітних спеціалізацій: один або декілька продюсерів для нагляду за виробництвом, гейм дизайнери, художники, сценаристи, звукооператори (композитори, творці звукових ефектів), тестувальники.

Визначено етапи створення графічних ігор: розробка ідеї, визначення середі розробки, написання коду, впровадження графічних та аудіо елементів тощо, процес доопрацювання гри, використання та продаж гри за допомогою систем по продажі програм.

Встановлено, що принцип кросплатформного підходу, який полягає у властивості програмного забезпечення працювати більш ніж на одній програмній або апаратній платформі, дозволяє суттєво зменшити витрати у майбутньому при необхідності перенести гру з однієї платформи на іншу.

Забезпечити принцип кросплатформності можна за допомогою LibGDX – Java фреймворка, який має всі засоби для розробки 2D та 3D-ігор.

РОЗДІЛ 2.

РОЗРОБКА КРОСПЛАТФОРМНОЇ ГРИ SIEGE ЗА ДОПОМОГОЮ ФРЕЙМВОРКУ LIBGDX

2.1. Підготовчий етап розробки програмного додатку

На початкуковому етапі було розроблено сценарій та правила гри Siege. Цей крок дозволив визначити головні об'єкти гри, логіку їх дій та умови перемоги або програшу.

Розробка сценарію гри Siege

За жанром гра Siege – це аркада. У даній грі ми граємо за головного героя Піроманта який повинен звільнити дев'ять рівнів від ворогів, щоб дістатися до Мармурового палацу.

Піромант розпочинає свій шлях на першому рівні у центрі екрану. У якості зброї нам доступні лише вогняні шари. Вороги з'являються за межами екрану та переміщуються до головного героя, щоб зупинити його.

Перехід на наступний рівень можливий тільки після того, як Піромант перемає n-ну кількість ворогів на даному рівні. Між рівнями герой зупиняється у безпечній зоні, де він відновлює свої сили. Щоб перейти звідси до наступного рівня необхідно спуститися по мотузці. На останньому рівні Піроманта чекатиме битва з босом. Після перемоги над босом Піромант дістане ця до Мармурового палацу.

Вибір інструментів розробки крос платформної гри Siege

Гру Siege будемо розробляти у двох версіях: для смартфона під управлінням операційної системи Android та для усіх персональних комп'ютерів, які підтримують Java. Ми вже вирішили, що для кросплатформності нашої графічної гри ми використовуватимемо фреймворк LibGDX. LibGdX проєкти використовують систему автоматичної збірки проєктів Gradle для керування залежностями, процесом зборки та інтеграцією з середовищем розробки. Це дозволяє розробляти додатки у будь-якому середовищі розробки. Серед безкоштовних середовищ розробки додатків ми обрали AndroidStudio. Ця програма дозволяє легко створити android-додаток а з фреймворком LibGDX

можна створювати програми як для смартфонів так і для персональних комп'ютерів.

2.2. Архітектура проєкту (огляд класів та об'єктів)

Проєкт LibGDX має жорстку структуру. LibGDX проєкт створюється з декількома класами. Один з цих класів – клас `Siege.java`, з яким будуть пов'язані усі створені класи та об'єкти. Виходячи з цього ми будемо розробляти гру за принципами об'єктно-орієнтованого програмування, які розглядають програму як множину «об'єктів», що взаємодіють між собою.

Проаналізувавши сценарій, було виявлено, що для гри необхідні головне екран (меню), головний герой, вороги, рівні та «сцена», де будуть взаємодіяти персонаж, вороги та рівні. У додатку А зображено UML діаграму класів гри `Siege`. На цьому рисунку стрілками зображено, де створюється об'єкт класу окрім двох випадків, де відбувається наслідування.

У `MainMenuScreen.java` буде відображатися головне меню гри, звідки можна розпочати гру, обрати альтернативний спосіб керування персонажем, якщо гра запущена на смартфоні, подивитися на коротку допомогу гравцю або вийти з гри. Довідка по грі буде доступна у класі `Help.java`.

У `StageScreen.java` гравець буде керувати персонажем та відбиватиметься від ворогів, які йтимуть на нього. У цьому класі будуть реалізовані персонаж, його зброя, вороги та рівні.

Для покращення відображення на екрані смартфонів буде зроблено абстрактний клас `GameCamera.java`, у якому створимо параметри для оптимального відображення.

Опис характеристик персонажа буде проводитись у класі `Player.java`, де буде створені його анімація, швидкість переміщення та способи керування ним.

Опис зброї персонажа проводитиметься у класі `FireBall.java`, де будуть створені його характеристики такі як потужність, кількість, швидкість польоту снарядів тощо.

Основні параметри та змінні для ворогів будуть створені у абстрактному класі `Enemies.java`. Кількість видів ворогів дорівнювати кількості рівнів – тобто

дев'ять. Усі вони наслідуватимуть параметри від абстрактного класу Enemies.java.

2.3. Розробка інтерфейсу та «сцени» гри

Абстрактний клас GameCamera

Для роботи з відображенням у LibGDX є клас OrthographicCamera. Цей клас працює як дуже проста камера реального миру. Цей клас допоможе нам дивитися на ігровий процес ніби через «вікно». Завдяки цьому на смартфонах гра буде на весь екран, а не на його частині. Код абстрактного класу наведено у додатку Б.

Стартовий екран (меню)

У більшості відеоігор є екран головного меню, і наша гра не виняток. У нашого екрану повинні бути точка входу у гру або вихід з неї, можливість змінити спосіб керування головним героєм та можливість побачити довідку гравця.

Клас стартового екрану має назву MainMenuScreen.java. Він наслідує абстрактний клас GameCamera.java для покращення відображення на смартфонах.

Перечислимо класи які надає нам LibGDX які ми використовували для створення меню гри: Texture, TextureRegion, TextureRegionDrawable, Music, Vector2, Stage, ImageButton, BitmapFont.

Об'єкти класу Texture отримують зображення з файлу, і завантажують його в GPU. Об'єкти класу описують прямокутник всередині текстури та використовуються для малювання тільки частини текстури.

Об'єкти класу Music отримують аудіо з файлу. Цей клас використовується тоді, коли необхідно програвати аудіо файл більш ніж хвилину.

Об'єкти класу Vector2 використовуються як спрощення у визначенні x та y у системі координат. Усі графічні об'єкти гри розміщуються за Декартовою системою координат початок відліку якого починається у лівому нижньому кутку екрану.

Об'єкт класу Stage допоможе у створенні інтерфейсу з яким ми будемо взаємодіяти у грі. ImageButton - з англ. дослівно зображення, кнопка. Цей клас допоможе створити інтерфейс, а саме кнопки з якими ми взаємодіятимемо.

Клас BitmapFont дозволяє відображувати на екрані будь-який текст.

Код класу MainMenuScreen.java наведено у додатку В.

У меню гри чотири інтерактивні кнопки: NewGame – розпочати гру, Joystic – обрати спосіб керування головним героєм, Help – подивитися довідку гравця, Exit – вихід з гри. Під час активності стартового екрану гратиме музика. Готовий інтерфейс екрану меню зображено на рис 2.1.

«Сцені» гри

На «сцені» відбуваються головні події гри: керування головним персонажем, напад ворогів на нього, очищення рівнів, тощо. Для цієї «сцени» необхідні задній фон, фонові музика, два джойстика для управління через сенсорний екран параметри для відображення деякої інформації на екрані (кількість здоров'я головного героя, набраних очок тощо). Фон був організований з використанням класу Texture, музика – використанням Music, а джойстик – використанням декількох класів: Touchpad, TouchpadStyle, Skin та Drawable.



Рис 2.1. Готовий інтерфейс екрану меню

Touchpad – екранний джойстик. Область руху джойстика має круглу форму, з центром на сенсорній панелі, і його розмір обмежується меншим колом. Розмір тачпаду визначається його зображенням заднього фону.

TouchpadSkin використовується для налаштування зовнішнього вигляду джойстика. Клас Drawable тут використовується для відображення джойстику на

екрані. Лістинг коду StageScreen.java наведено у додатку Д. «Сцену» гри зображено на рис 2.2.

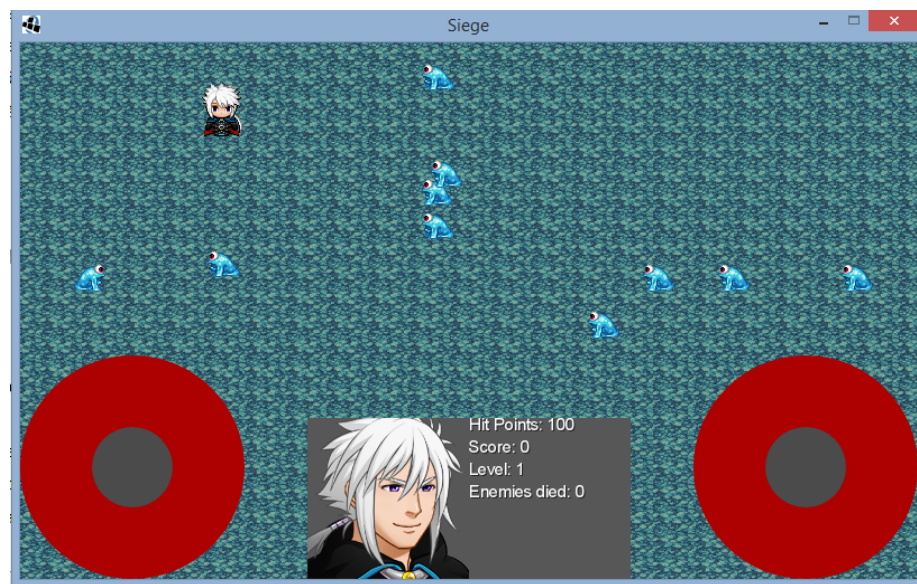


Рис 2.2. «Сцена» гри з готовим персонажем та ворогами

2.4. Розробка об'єктів персонажів гри

Розробка об'єкту головного героя

Клас головного героя має назву Player.java. Для створення зображення головного героя було використано генератор персонажів, який є частиною програми RPGMakerMV. Для головного героя були створені такі основні параметри як кількість здоров'я, сила пошкодження, отримання пошкодження, швидкість пересування.

Для анімації переміщення були використані класи Animation, Texture, TextureRegion та SpriteBatch. Клас Animation допомагає зробити анімацію з зображень, що зберігаються у GPU. Для цього текстура «ріжеться» на декілька кадрів, ці кадри поміщуються у масив, а з цього масиву клас Animation підмальовує на екрані ці зображення по чергово.

Дуже часто малювання текстур відбувається в прямокутній геометрії. Також дуже часто одна текстура або її різні частини малюються багато разів. Неefективно відсилати один прямокутник для відтворення в GPU. Замість цього, багато прямокутників для однієї текстури можуть бути описані і відправлені в GPU усі разом. Цим і займається клас SpriteBatch.

Також для цього класу було використано такий клас як Sound. Цей клас використовується для програвання коротких аудіо. У якості аудіо було обрано звук отримання пошкоджень головним героєм.

Лістинг коду Player.java наведено у додатку Е.

Розробка зброї для головного героя

У якості зброї головний герой використовуватиме вогняні шари, якими він може керувати. Цей клас має назву FireBall.java. Для анімації польоту вогняного шара використовувалися ті ж засоби, що й при анімації головного героя. Також були додані такі параметри як швидкість польоту та активність вогняного шара. Лістинг коду FireBall.java наведено у додатку Ж. Дію вогняних шарів зображено на рис 2.3.



Рис 2.3. Вогняні шари у дії

Розробка об'єктів ворогів

Щоб зменшити кількість коду у класах ворогів був розроблен абстрактний клас Enemies.java у якому містяться основні класи та параметри (кількість здоров'я, швидкість переміщення, сила пошкодження, кількість нарахованих очок тощо), які необхідні для анімації ворогів та їхньої логіки. Лістинг абстрактного класу наведено у додатку З. У якості прикладу класа ворога було обрано клас EnemySlime.java лістинг якого наведено у додатку К.

2.5. Додавання створених персонажів до «сцени» та робота над логікою гри

Для того, щоб додати створених персонажів до «сцени» у якості об'єктів їх необхідно оголосити як клас у StageScreen.java та створити їхні об'єкти у конструкторі StageScreen.

У «сцені» було створено дванадцять текстур з поміж яких дев'ять – це текстур заднього фону локацій, одна текстура – безпечна зона, одна текстура – зона перемоги та одна текстура – задній фон екрану програшу.

Головний герой розміщується у центрі локації за допомогою параметра positionOfPlayer класу Vector2, який приймає половину значення розміру екрану гри по x та по y . Переміщення Піроманта по локації відбувається за допомогою кнопок клавіатури WASD, або за допомогою лівого джойстика, якщо він увімкнений. Атака відбувається за допомогою стрілок клавіатури $\uparrow\downarrow\leftarrow\rightarrow$, або за допомогою правого джойстика.

Вороги розміщуються за межою екрану. Для цього було використано генератор чисел у діапазоні від 0 до 16. Значення від 0 до 4 – вороги розміщуються зправа від екрану, значення від 5 до 8 – вони розміщуються знизу від екрану, від 9 до 12 – розміщуються ліворуч від екрану, від 13 до 16 – розміщуються зверху екрану. Якщо ворог був знищений, він з'являється у случайній частині за екраном згідно з алгоритмом їхнього розміщення.

Для створення колізії між персонажем та ворогами для кожного з них були створені хіт-бокси – прозорі прямокутники розміром з один спрайт головного героя або ворога. Ці хіт-бокси переміщуються разом з тими, для кого вони були зроблені. При зіткненні головного персонажа з ворогом, Піромант отримує поранення, яке дорівнює силі ворога та переміщується у довільну частину екрану. Коли у героя закінчується здоров'я, на екрані з'являється повідомлення про програш, потім гра завершується. При зіткненні вогняного шара з ворогом, у ворога зменшується рівень здоров'я. Як тільки здоров'я впаде до нуля або нижче, ворог знищиться й зникне з екрану. Як тільки це відбудеться, гравець отримує бали згідно з вартістю ворога.

Перехід на наступні локації відбувається тільки тоді, коли буде знищено п-на кількість ворогів. Між локаціями з ворогами герой перебуває у безпечній зоні, де його здоров'я відновлюється. Для виходу з цієї локації необхідно спуститися по мотузці.

Для преміщення ворогів до Піроманта використовуються їхні координати x та y .

Для виходу з гри є три шляхи. Перший – закрити гру зручним чином.. Другий – програти та натиснути на екран. Третій – це пройти гру. На останньому рівні у Мармуровому палаці необхідно стати у верхній частині екрану посередині.

2.6. Реалізація кросплатформності

Як вже говорилося, LibGDX проєкт вже створюється з декількома класами. Їхня кількість залежить від того, для яких платформ розробляється додаток. Гра «Siege» розробляється для ПК та смартфонів на базі ОС Android. Для того, щоб гра працювала на цих двох платформах у LibGDX проєкті було створено такі два класи як DesktopLauncher.java, завдяки якому гру можливо запустити на ПК та AndroidLauncher.java, завдяки якому гру можливо запустити на смартфонах на базі ОС Android.

У DesktopLauncher.java імпортуються дві LibGDX бібліотеки, завдяки яким гра буде працювати на ПК, та клас Siege.java, який є головним класом даного проєкту. У конструкторі класу можливо налаштувати параметри вікна гри: назву вікна, його розмір, чи можливо змінити розмір вікна, чи буде екран у режим fullscreen тощо. Також у конструкторі ініціалізується запуск гри на ПК. Лістинг коду класу DesktopLauncher наведено нижче:

```
public class DesktopLauncher {  
    public static void main (String[] arg) {  
        LwjglApplicationConfiguration config = new  
LwjglApplicationConfiguration();  
        config.title = Siege.TITLE;  
        config.width = Siege.WIDTH;  
        config.height = Siege.HEIGHT;  
        config.resizable = false;  
        //config.fullscreen = false;  
        //config.fullscreen = true;  
        new LwjglApplication(new Siege(), config);  
    }  
}
```

У `AndroidLauncher.java` імпортуються дві бібліотеки `LibGDX`, завдяки яким гра буде працювати на ОС Android, та клас `Siege.java`. У конструкторі класу можливо налаштувати такі параметри як: використання компасу, акселерометру, GPS, орієнтація екрану тощо. Також у цьому конструкторі ініціалізується запуск гри для смартфонів.

Лістинг коду класу `AndroidLauncher` наведено нижче.

```
public class AndroidLauncher extends AndroidApplication {  
    @Override  
    protected void onCreate (Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        AndroidApplicationConfiguration config = new  
        AndroidApplicationConfiguration();  
        config.useAccelerometer = false;  
        config.useCompass = false;  
        initialize(new Siege(), config);  
    }  
}
```

На рис. 2.4 та 2.5 зображено емулятор смартфона з запущеною грою.



Рис. 2.4. Головне меню гри запущеної у емуляторі Android

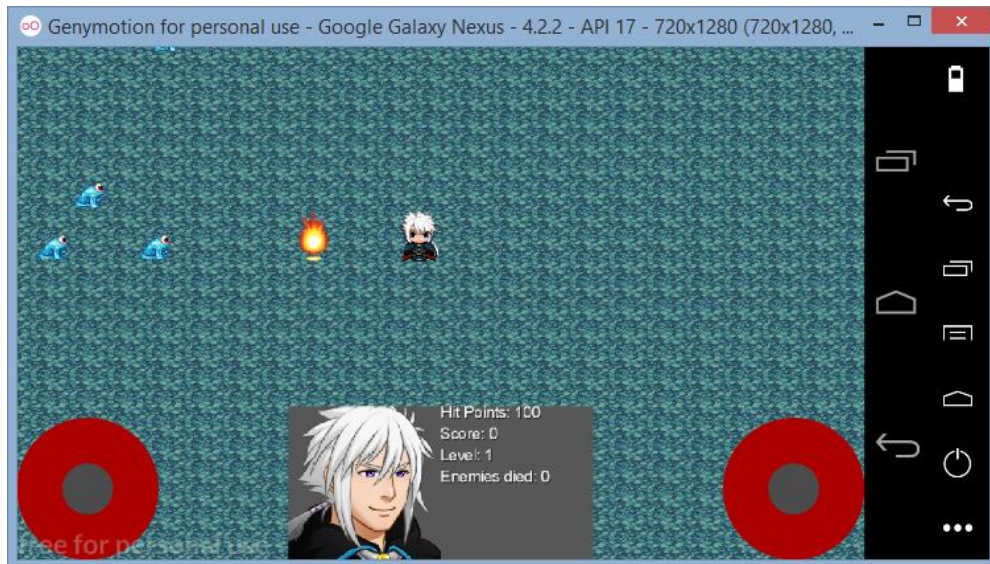


Рис. 2.5. «Сцена» гри, яка запущена у емуляторі Android

Таким чином, гра повністю демонструє свої функції й можливості на різних платформах ПК та ОС Android.

2.7. Керівництво користувача

Підготовка до роботи з додатком

Персональні комп'ютери

Кросплатформна графічна гра «Siege» була розроблена мовою програмування Java. Тому для того щоб на ПК було можливо запустити цю гру необхідна наявність програмного забезпечення Java. Якщо на вашому ПК це ПЗ не встановлено, його можливо завантажити з офіційного сайту Oracle Corporation [US] java.com/ru/download/. Для того, щоб його встановити треба запустити завантажений файл та слідувати вказівкам.

Смартфони на базі ОС Android

Єдині технічні умови для цієї платформи – це те, що на смартфонах повина бути встановлена ОС Android версії 4.4. Якщо ви не знаєте версію вашого ОС Android, то його можливо подивитися у налаштуваннях смартфонів у графі «О телефоні». Якщо версія нижче необхідної, то треба подивитися в Інтернеті чи підтримує ваш смартфон дану версію ОС. Якщо так, то завантажити нову версію Android можливо декількома способами. Перший спосіб – це «повітряне завантаження», тобто за допомогою wifi у налаштуваннях телефону в графі «О телефоні» можливо завантажити нову версію, слідуючи вказівкам телефону. Другий спосіб – це завантажити до смартфона нову версію ОС за допомогою

комп'ютера. Для цього є спеціальні програми, які зазвичай йдуть у комплекті з телефоном. Запустіть програму на ПК, потім підключіть телефон до нього через USB кабель. Потім у програмі оберіть «Завантажити нову версію ОС Android». Далі слідкувати вказівкам програми.

Робота з додатком

Робота на ПК

Взаємодія між інтерфейсом та користувачем здійснюється за допомогою мишки.

Керування головним героєм проводиться через клавіатуру, а саме через клавіші **WASD**. Може бути, що головний герой не переміщується. Це пов'язано з тим, що розкладка клавіатури не співпадає з ігровими на лаштунками. Для того щоб це виправити необхідно увімкнути «англійську» розкладку клавіатури.

Керування вогнем проводиться через клавіатуру, а саме клавішами →←↓↑.

Гру, запущену на ПК, зображено на Рис.2.6.

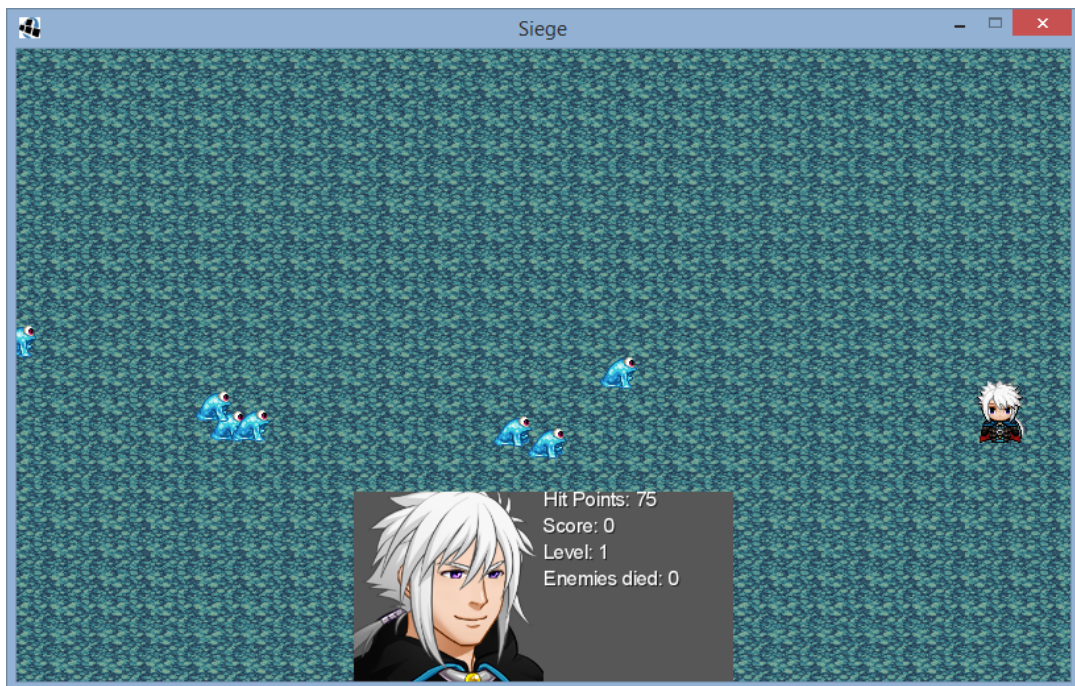


Рис. 2.6. Гра «Siege» запущена на ПК

Робота на смартфоні

Взаємодія між інтерфейсом та користувачем здійснюється через екран смартфона, тобто через дотик.

Для керування головним героєм та вогнем у грі передбачена можливість увімкнути сенсорні джойстики у головному меню. Лівий джойстик відповідає за керування героєм, правий – за керування вогнем.

Гру, запущену на емуляторі смартфоні, зображено на Рис. 2.7.

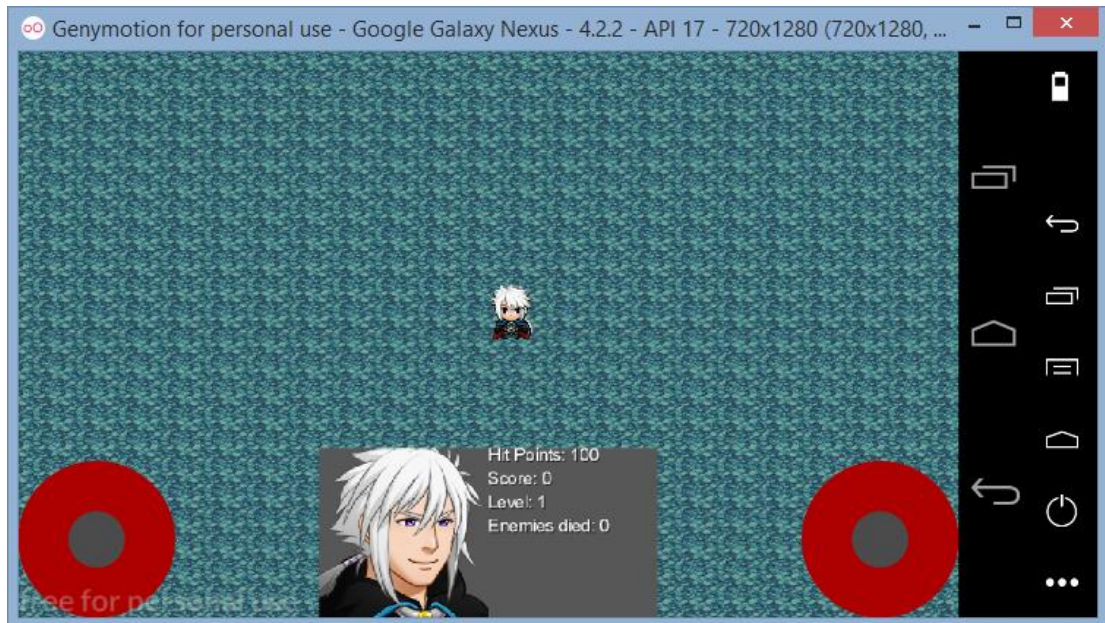


Рис. 2.7. Гра «Siege», на емуляторі смартфона

Ігровий процес

Для того щоб переходити на наступні рівні необхідно перемогти певну кількість ворогів.

Вороги йтимуть до героя з певною швидкістю й коли вони зіткнуться, герой отримує пошкодження та переміститься у довільну частину екрану. Коли кількість здоров'я героя дійде до 0, гра покаже екран програшу та вимкнеться.

Коли кількість здоров'я ворогів дійде до 0, вони зникають, а нові з'являються за межами екрану, головний герой отримує n-ну кількість балів, яка дорівнює вартості переможеного ворога.

Гра завершиться, коли герой переможе боса та потрапить у Мармуровий палац (а саме у середину верхньої частини екрану).

Висновки до розділу

У другому розділі кваліфікаційної роботи було створено проєкт графічної гри Siege засобами фреймворку LibGDX, розроблені класи стартового екрану гри, головної «сцени», де відбуваються основні події, головного героя, який у якості зброї використовує вогняні шари, дев'ять типів ворогів та логіку гри.

Створені дві версії гри: версія для ПК та для смартфона на базі операційної системи Android.

Таким чином, розроблено графічну гру Siege, яка є кросплатформною, і відповідає усім вимогам.

ВИСНОВКИ

У ході кваліфікаційної роботи розроблено кросплатформну 2D гру засобами JAVA та LIBGDX. Під час розробки розглянуто методологічні підходи до розробки графічних ігрових додатків, а саме визначено класифікаційні підходи:

- за жанрами: Action, симулятор/менеджмент, стратегії, пригоди, музичні ігри, рольові ігри (RPG), головоломки/ логічні/ пазли, традиційні настільні ігри, текстові ігри.
- за кількістю гравців: одно та багато користувальні, багато користувальні на одному комп'ютері, багато користувальні офлайн-ігри, масові онлайн ігри.
- за стилістикою: вестерн, кіберпанк, космічні, постапокаліптичні, стімпанк, сучасні, фентезі;
- за платформою ігри поділяються на персональні комп'ютери, ігрові консолі / приставки, мобільні телефони і КПК.

Обґрунтовано професійний напрям спеціалістів, що задіяні у створенні гри, та ключові етапи розробки:

- розробка ідеї;
- визначення середовища розробки;
- написання коду;
- впровадження графічних та аудіо елементів тощо;
- процес доопрацювання гри;
- використання та продаж гри.

Встановлено, що принцип кросплатформного підходу дозволяє суттєво зменшити витрати у майбутньому при необхідності перенести гру з однієї платформи на іншу. Забезпечити принцип кросплатформності можна за допомогою LibGDX –Java фреймворка.

- Під час розробки проєкту гри Siege були створені сценарій та правила гри, класи екрану меню та «сцени», класи та об'єкти персонажів. Була розроблена логіка гри.
- Таким чином, розроблено графічну кросплатформну гру Siege, яка

відповідає усім встановленим вимогам.

- Розроблений проєкт може застосовуватися у навчальних цілях під час підготовки відповідних напрямків спеціалістів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Галёнкин Сергей. Маркетинг игр / Сергей Галёнкин. – 2014. – 78 с.
2. Global mobile OS market share in sales to end users from 1st quarter 2009 to 1st quarter 2016 [Электронный ресурс] / 2016. – Режим доступа: <https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems>.
3. New mobile game statistics every game publisher should know in 2016 [Электронный ресурс] / 2016. – Режим доступа: <https://www.surveymonkey.com/business/intelligence/mobile-game-statistics>.
4. Scott Robertson. How to design: Concept Design Process, Styling, Inspiration, and Methodology. – Titan Books, 2014. – 176 с.
5. Andrew M. Understanding Open Source and Free Software Licensing / Andrew M. St. Laurent. – O'Reilly Media, 2004. – 207 с.
6. About CMake [Электронный ресурс]. Режим доступа: <https://cmake.org/overview>.
7. Engelbert Roger. Cocos2d-x by Example Beginner's Guide / Roger Engelbert. – Packt Publishing Ltd, 2013. – 246 с.
8. Shah Ryan. Blueprints Master the Art of Unreal Engine 4 – Blueprints / Ryan Shah. – CreateSpace Independent Publishing Platform, 2014. – 122 с.
9. ISO/IEC 14882:2014. Information technology – Programming languages – C++ / ISO, 2014. – Режим доступа: <https://www.iso.org/standard/64029.html>.
10. Интегрированная среда разработки Visual Studio [Электронный ресурс] / Microsoft, 2017. – Режим доступа: <https://www.visualstudio.com/ru/vs/>.
11. Bourg David M. Physics for Game Developers, 2nd Edition / David M Bourg, Bryan Bywalec. – O'Reilly Media, 2013. – 578 с.
12. Korth Andy. Chipmunk 7 released – Pro tools open sourced [Электронный ресурс] / Andy Korth. – Howling Moon Software, 2015. – Режим доступа: <http://howlingmoonsoftware.com/wordpress/chipmunk-7-released-pro-tools-open-sourced>.

13. Feronato Emanuele. Box2D for Flash Games / Emanuele Feronato. – Packt Publishing, 2012. – 166 с.
14. Parberry I. Introduction to Game Physics with Box2D / Ian Parberry. – CRC Press, 2013. – 275 с.
15. Scott Chacon. Pro Git [Электронный ресурс] / Scott Chacon, Ben Straub. – 2014. – Режим доступа: <https://git-scm.com/book/en/v2>.
16. Чакон Скотт. Git для профессионального программиста / Скотт Чакон, Бен Штрауб. – Питер, 2016. – 496 с.
17. Фримен Элизабет. Паттерны проектирования / [Элизабет Фримен, Эрик Фримен, Кэти Сиерра, Берт Бейтс]. – Питер, 2013. – 656 с.
18. Смит Джейсон Мак-Колм. Элементарные шаблоны проектирования / Джейсон Мак-Колм Смит. – М.: «Вильямс», 2012. – 304 с.
19. Гамма Эрих. Приемы объектно-ориентированного проектирования. Паттерны проектирования / [Эрих Гамма, Ричард Хелм, Ральф Джонсон, Джон Влссидес]. – Питер, 2016. – 366 с.
20. Макконнелл Стив. Совершенный код. Мастер-класс / Стив Макконнелл. – Русская Редакция, 2017. – 896 с.
21. Scott Meyers. Effective Modern C++ / Scott Meyers. – 2014. – 334 с.
22. *OMG. OMG Unified Modeling Language (OMG UML), Superstructure Version 2.5 / OMG, 2015.* – Режим доступа: <http://www.omg.org/spec/UML/2.5/PDF>.
23. Despain. 100 Principles of Game Design / Despain, Wendy. – New Riders, 2012. – 240 с.
24. Erin Catto. World Class. Simulation [Электронный ресурс] / Erin Catto // Box2D v2.3.0 User Manual. – 2013. – Режим доступа: <http://box2d.org/manual.pdf>.
25. Fiedler Glenn. Fix Your Timestep! [Электронный ресурс] / Glenn Fiedler. – 2016. – Режим доступа: <http://gafferongames.com/game-physics/fix-your-timestep>.
26. O'Haver T. Smoothing [Электронный ресурс] / O'Haver T. – 2012. – Режим доступа:

<http://terpconnect.umd.edu/~toh/spectrum/Smoothing.html>.

27. Corless Robert. A Graduate Introduction to Numerical Methods – From the Viewpoint of Backward Error Analysis / Robert Corless, Nicolas Fillion. – Springer Science & Business Media, 2013. – 869 с.
28. Abel Gomes. Geometric Computing. Lecture 4 – Interpolation methods [Электронный ресурс] / Abel Gomes. – 2010. – Режим доступа: <http://www.di.ubi.pt/~agomes/tcg/lectures/04-lecture.pdf>.
29. Matsuura Akihiro. Cocos2d-x Cookbook / Akihiro Matsuura. – UK: Packt Publishing, 2015. – 250 с.
30. Siddharth Shekar. Cocos2d Cross-Platform Game Development Cookbook – Second Edition / Shekar Siddharth. – UK: Packt Publishing, 2016. – 384 с.
31. Hernandez Raydelto. Building Android Games with Cocos2d-x / Raydelto Hernandez. – UK: Packt Publishing, 2015. – 147 с.
32. *Kehoe Donald*. Designing Artificial Intelligence for Games [Электронный ресурс] / *Donald Kehoe*. – 2015. – Режим доступа: <https://software.intel.com/en-us/articles/designing-artificial-intelligence-for-games-part-1>.
33. Heaton Jeff. Artificial Intelligence for Humans, Volume 1: Fundamental Algorithms / Jeff Heaton. – Heaton Research, 2013. – 147 с.
34. Aung Sithu Kyaw. Unity 4.x Game AI Programming / Aung Sithu Kyaw, Clifford Peters, Thet Naing Swe. – Packt Publishing, 2013. – 232 с.
35. Rabin Steven. Game AI Pro: Collected Wisdom of Game AI Professionals / Steven Rabin. – A K Peters/CRC Press, 2013. – 626 с.
36. Bevilacqua Fernando. Understanding Steering Behaviors [Электронный ресурс] / Fernando Bevilacqua. – 2013. – Режим доступа: <https://gamedevelopment.tutsplus.com/tutorials/understanding-steering-behaviors-movement-manager--gamedev-4278>.

Додаток А. Ієрархія класів кросплатформної гри Siege

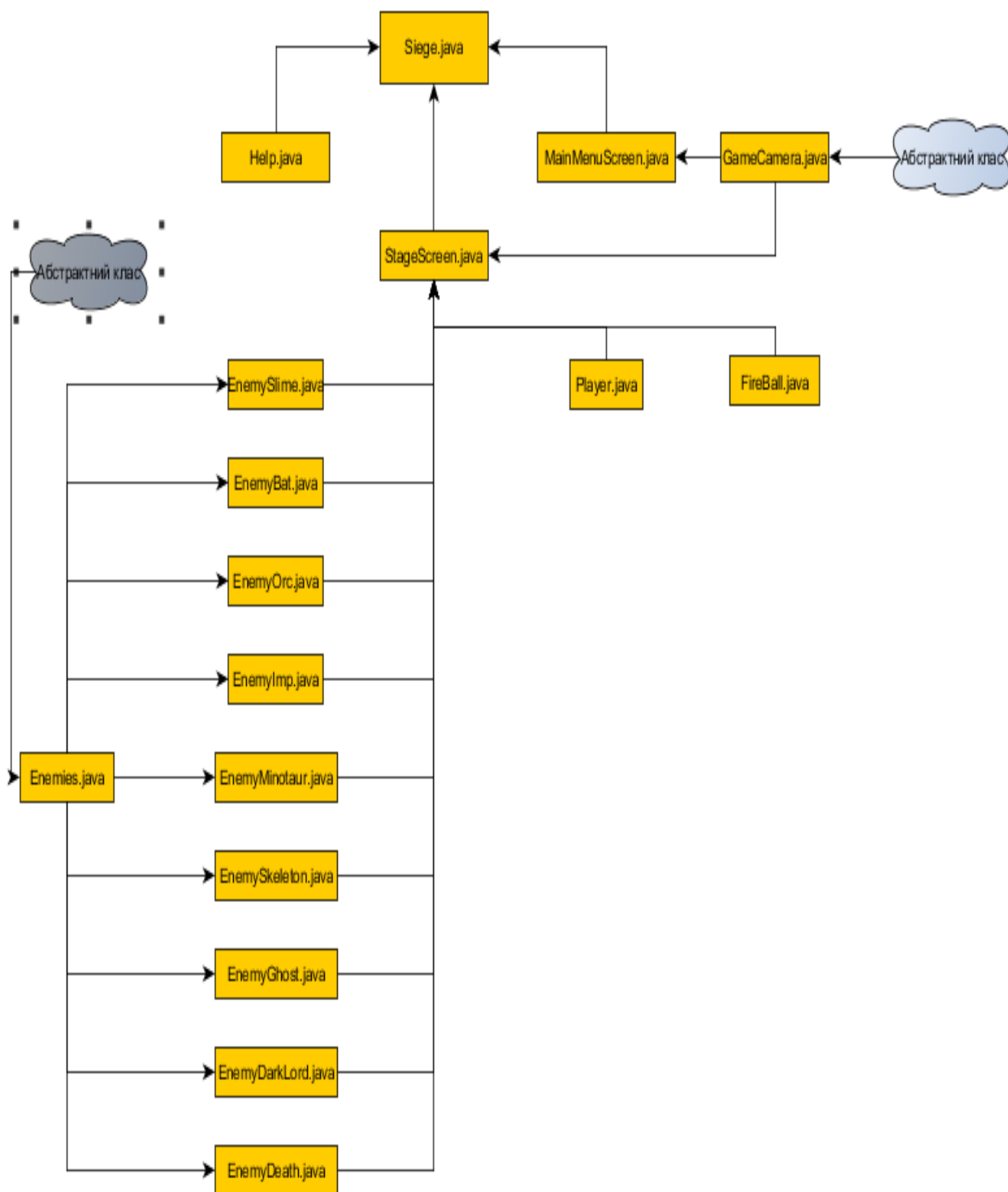


Рис.А.1. UML діаграма класів та об'єктів кросплатформної гри Siege

Додаток Б. Лістинг абстрактного класу GameCamera.java

```
package com.crossgame.siege;

import com.badlogic.gdx.graphics.OrthographicCamera;
import com.badlogic.gdx.math.Vector3;
/**
 * Created by Fenrir on 20.02.2017.
 */
public abstract class GameCamera {

    protected OrthographicCamera camera;
    protected Vector3 mouse;

    public GameCamera(){
        camera = new OrthographicCamera ( );
        mouse = new Vector3 ( );
    }

    protected abstract void handleInput();
}
```

Додаток В. Лістинг стартового екрану (меню) MainMenuScreen.java

```
package com.crossgame.siege.GSM;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.audio.Music;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.BitmapFont;
import com.badlogic.gdx.graphics.g2d.SpriteBatch;
import com.badlogic.gdx.graphics.g2d.TextureRegion;
import com.badlogic.gdx.math.Rectangle;
import com.badlogic.gdx.math.Vector2;
import com.badlogic.gdx.scenes.scene2d.InputEvent;
import com.badlogic.gdx.scenes.scene2d.Stage;
import com.badlogic.gdx.scenes.scene2d.ui.ImageButton;
import com.badlogic.gdx.scenes.scene2d.utils.ClickListener;
import com.badlogic.gdx.scenes.scene2d.utils.TextureRegionDrawable;
import com.badlogic.gdx.utils.viewport.ScreenViewport;
import com.crossgame.siege.GameCamera;
import com.crossgame.siege.Siege;
import com.crossgame.siege.StageScreen;

/**
 * Created by Fenriron 22.01.2017.
 */

public class MainMenuScreen extends GameCamera {

    /**Главное меню*/
    private boolean mainActive; //Для управления активностью главного меню

    public boolean isMainActive () {
        return mainActive;
    }

    public void setMainActive ( boolean mainActive ) {
        this.mainActive = mainActive;
    }

    private Music themeMusic; //Фоновая музыка главного меню

    public Music getThemeMusic () {
        return themeMusic;
    }

    private Texture texMenuBackground; //Для текстуры заднего фона

    private Vector2 posNewGameButton;
    private Vector2 posJoysicActive;
    private Vector2 posHelpButton;
    private Vector2 posExitButton;

    private Stage menuStage;
    public Stage getMenuStage () {
        return menuStage;
    }
}
```

```
}
```

```
private ImageButton newGameButton;  
private ImageButton joystickActiveButton;  
private ImageButton helpButton;  
private ImageButton exitButton;  
private ImageButton goToMenu;  
private ImageButton on;  
private ImageButton off;
```

```
private TextureRegion regNewGame;  
private TextureRegion regJoystickActive;  
private TextureRegion regHelp;  
private TextureRegion regExit;  
private TextureRegion regOn;  
private TextureRegion regOff;
```

```
private TextureRegionDrawable drNewGame;  
private TextureRegionDrawable drJoystickActive;  
private TextureRegionDrawable drHelp;  
private TextureRegionDrawable drExit;  
private TextureRegionDrawable drOn;  
private TextureRegionDrawable drOff;
```

```
//Длятекстуркнопок
```

```
private static Texture texNewGameButton;  
private static Texture texJoystickActive;  
private static Texture texHelpButton;  
private static Texture texExitButton;  
private static Texture On;  
private static Texture Off;
```

```
/**Включение/выключениеджойстика*/
```

```
private boolean joystickActive;  
public boolean isJoystickActive(){  
    return joystickActive;  
}
```

```
private BitmapFont joystickOn;  
private BitmapFont joystickOff;
```

```
/**Помощь*/
```

```
private boolean helpActive;
```

```
public boolean isHelpActive () {  
    return helpActive;  
}
```

```
public void setHelpActive ( boolean helpActive ) {  
    this.helpActive = helpActive;  
}
```

```
private BitmapFont help;
```

```

public MainMenuScreen() {

/**Меню*/
mainActive = true;
    helpActive = false;

    themeMusic = Gdx.audio.newMusic ( Gdx.files.internal ( "Audio/bgm/Theme1.ogg" ) );
    themeMusic.setLooping ( true );

    camera.setToOrtho ( false, Siege.WIDTH, Siege.HEIGHT );
    //camera.setToOrtho ( false, Gdx.graphics.getWidth (), Gdx.graphics.getHeight () );

    //Загружаемзаднийфон
    texMenuBackground = new Texture("Backgrounds/Menu.png");

    menuStage = new Stage ( new ScreenViewport () );

    //Загружаемтекстурыкнопок
    texNewGameButton = new Texture(Gdx.files.internal ( "ActionButtons/NewGame.png"));
    texJoysticActive = new Texture(Gdx.files.internal ( "ActionButtons/Joystics.png"));
    texHelpButton = new Texture ( Gdx.files.internal ( "ActionButtons/Help.png" ));
    texExitButton = new Texture(Gdx.files.internal ( "ActionButtons/Exit.png"));

    On = new Texture ( Gdx.files.internal ( "ActionButtons/On.png" ) );
    Off = new Texture ( Gdx.files.internal ( "ActionButtons/Off.png" ) );

    regNewGame = new TextureRegion ( texNewGameButton );
    regJoysticActive = new TextureRegion ( texJoysticActive );
    regHelp = new TextureRegion ( texHelpButton );
    regExit = new TextureRegion ( texExitButton );
    regOn = new TextureRegion ( On );
    regOff = new TextureRegion ( Off );

    drNewGame = new TextureRegionDrawable ( regNewGame );
    drJoysticActive = new TextureRegionDrawable ( regJoysticActive );
    drHelp = new TextureRegionDrawable ( regHelp );
    drExit = new TextureRegionDrawable ( regExit );
    drOn = new TextureRegionDrawable ( regOn );
    drOff = new TextureRegionDrawable ( regOff );

    posNewGameButton = new Vector2 ( Gdx.graphics.getWidth ()/2 -
(texNewGameButton.getWidth ()/2), 350 );
    posJoysicActive = new Vector2 ( Gdx.graphics.getWidth ()/2 - (texJoysticActive.getWidth
()/2), 250 );
    posHelpButton = new Vector2 ( Gdx.graphics.getWidth ()/2 - (texHelpButton.getWidth ()/2),
150 );
    posExitButton = new Vector2 ( Gdx.graphics.getWidth ()/2 - (texExitButton.getWidth ()/2), 50
);

    newGameButton = new ImageButton ( drNewGame );
    joysticAvtiveButton = new ImageButton ( drJoysticActive );
    helpButton = new ImageButton ( drHelp );
    exitButton = new ImageButton ( drExit );

```



```

on = new ImageButton ( drOn );
off = new ImageButton ( drOff );

joysticActive = false;

newGameButton.setPosition ( posNewGameButton.x, posNewGameButton.y );
joysticAvtiveButton.setPosition ( posJoysicActive.x, posJoysicActive.y );
helpButton.setPosition ( posHelpButton.x, posHelpButton.y );
exitButton.setPosition ( posExitButton.x, posExitButton.y );
on.setPosition ( posJoysicActive.x + texJoysticActive.getWidth (), 245 );
off.setPosition ( posJoysicActive.x + texJoysticActive.getWidth (), 245 );

if(mainActive) {
    menuStage.addActor ( newGameButton );
    menuStage.addActor ( joysticAvtiveButton );
    menuStage.addActor ( helpButton );
    menuStage.addActor ( exitButton );
/**
    menuStage.addActor ( on );
    menuStage.addActor ( off );
    */
}
    handleInput ();
}

//УправлениеВГлавномМеню

private void handleInputNewGame(){
    newGameButton.addListener ( new ClickListener ( ){
        public void clicked(InputEvent event, float x, float y){
            if(mainActive == true && !helpActive) {
                mainActive = false;
            }
        }
    } );
}

private void handleInputJoystic(){
    joysticAvtiveButton.addListener ( new ClickListener ( ){
        public void clicked(InputEvent event, float x, float y){
            if(mainActive == true && !helpActive) {
                if(!joysticActive) {
                    joysticActive = true;
                }else if(joysticActive && !helpActive){
                    joysticActive = false;
                }
            }
        }
    } );
}

private void handleInputHelp(){
    helpButton.addListener ( new ClickListener ( ){
        public void clicked(InputEvent event, float x, float y){

```

```

        if(mainActive && !helpActive) {
            helpActive = true;
        }
    }
});
}

private void handleInputExit(){
    exitButton.addListener ( new ClickListener ( ){
        public void clicked(InputEvent event, float x, float y){
            if(mainActive == true && !helpActive) {
                Gdx.app.exit ();
            }
        }
    } );
}

public void handleInput (){

    if(mainActive == true) {

        handleInputNewGame ();
        handleInputJoystic ();
        handleInputHelp ();
        handleInputExit ();
    }
}

public void joyactive(SpriteBatch batch){
    if(joysticActive){
        //batch.draw ( On, posJoysicActive.x + texJoysticActive.getWidth (), posJoysicActive.y -5
);
        menuStage.addActor ( on );
    }else if(!joysticActive){
        //batch.draw ( Off, posJoysicActive.x + texJoysticActive.getWidth (), posJoysicActive.y -5
);
        menuStage.addActor ( off );
    }
}
//else System.out.println("main is false");

public void render (SpriteBatch batch) {

    if(mainActive == true) {
        camera.update ();
        batch.draw ( texMenuBackground, 0, 0, Siege.WIDTH, Siege.HEIGHT );

        batch.setProjectionMatrix ( camera.combined );

        joyactive ( batch );
        //menuStage.draw ();
    }
}

```

```

    }

    public void stage(){
        menuStage.draw ();
    }

    public void update() {
        if(Gdx.input.justTouched ()){
            //System.out.println ("i = " + );
        }
        //handleInput ();
    }

    public void dispose() {
        themeMusic.dispose ();
        texMenuBackground.dispose ();
        texNewGameButton.dispose ();
        texJoysticActive.dispose ();
        texHelpButton.dispose ();
        texExitButton.dispose ();
        On.dispose ();
        Off.dispose ();
    }
}

```

Додаток Д. Лістинг частини коду «сцени» гри StageScreen.java

```
public class StageScreen extends GameCamera {

    //Музыка
    private Music battleMusic;//обычныйбой
    private Music bossBattleMusic;// бойсбосом
    private Music safeZoneMusic;//безопаснаязона
    private Music finishMusic;//победнаямузыка

    //Гетеры
    public Music getBattleMusic () {
        return battleMusic;
    }

    public Music getBossBattleMusic () {
        return bossBattleMusic;
    }

    public Music getSafeZoneMusic() {
        return safeZoneMusic;
    }

    public Music getFinishMusic() {
        return finishMusic;
    }

    //Кнопкиуправления

    private Stage stageScreenStage;//немного тафталогии)

    public Stage getStageScreenStage () {
        return stageScreenStage;
    }

    SpriteBatch batch;

    //Тачпад для передвижения игрока
    private Touchpad playerTouchpad;
    private TouchpadStyle playerTouchpadStyle;
    private Skin playerTouchpadSkin;
    private Drawable playerTouchBackground;
    private Drawable playerTouchKnob;

    //Тачпад для управления огненными шарами
    private Touchpad fireTouchpad;
    private TouchpadStyle fireTouchpadStyle;
    private Skin fireTouchpadSkin;
    private Drawable fireTouchBackground;
    private Drawable fireTouchKnob;

    private Vector2 middle;/**чтобы поместить игрока в центр экрана*/
    private Vector2 dispos;

    private Player player;
```

```

private static FireBall[] fireBalls;

private static EnemySlime[] slimes;
private static EnemyBat[] bats;
private static EnemyOrc[] orcs;
private static EnemyImp[] imps;
private static EnemyMinotaur[] minotaurs;
private static EnemySkeleton[] skeletons;
private static EnemyGhost[] ghosts;
private static EnemyDarkLord[] darkLords;
private static EnemyBossDeath[] deaths;

/**Длятекстанаэкрane*/
private ImageButton hud;
private Texture texHud;
private TextureRegion regHud;
private TextureRegionDrawable drHud;

private BitmapFont playerHP;
private BitmapFont score;
private BitmapFont level;
private BitmapFont enemiesDied;

private Vector2 posHud;
private Vector2 posPlayerHP;
private Vector2 posScore;
private Vector2 posLevel;
private Vector2 posEnemiesDied;

private int scoreCount;

public void setScoreCount ( int scoreCount ) {
    this.scoreCount = scoreCount;
}

private int levelCount;

public int getLevelCount () {
    return levelCount;
}

private int safeCount;
public int getSafeCount () {
    return safeCount;
}

private int enemiesDiedCount;

public void setEnemiesDiedCount ( int enemiesDiedCount ) {
    this.enemiesDiedCount = enemiesDiedCount;
}

/**Дляпереходанаследующиеуровни (сложности?)*/

```

```

private final int LEVEL2 = 1;
private int LEVEL3 = LEVEL2 + 1;
private int LEVEL4 = LEVEL3 + 1;
private int LEVEL5 = LEVEL4 + 1;
private int LEVEL6 = LEVEL5 + 1;
private int LEVEL7 = LEVEL6 + 1;
private int LEVEL8 = LEVEL7 + 1;
private int LEVEL9 = LEVEL8 + 1;
private int FINISH = LEVEL9 + 1;

private boolean stageActive;/**ДляуправленияактивностьюСцены*/
public boolean isStageActive () {
    return stageActive;
}

public void setStageActive ( boolean stageActive ) {
    this.stageActive = stageActive;
}

private boolean activateTouchpad;
public boolean isActivateTouchpad() { return activateTouchpad; }

public void setActivateTouchpad ( boolean activateTouchpad ) {
    this.activateTouchpad = activateTouchpad;
}

private boolean safe;
public boolean isSafe () { return safe; }

private boolean gameOver;
public boolean isGameOver(){
    return gameOver;
}

/**лятекстурзаднегофонасцены
 * котояделитсяна 9 уровней*/
private static Texture texSafeZone;
private static Texture texSlimeDungeon;
private static Texture texBatDungeon;
private static Texture texOrcDungeon;
private static Texture texImpDungeon;
private static Texture texMinotaurDungeon;
private static Texture texSkeletonDungeon;
private static Texture texGhostDungeon;
private static Texture texDarkLordDungeon;
private static Texture texDeathDungeon;
private static Texture texFinish;
private static Texture texGameOver;

private final int FIREBALL_COUNTS = 20;

private float fireballstart[];

/**ДлястрельбыОгненнымишарами*/

```

```

private int fireCounterRight;
    private int fireCounterLeft;
    private int fireCounterUp;
    private int fireCounterDown;
    private int fireRate;

/**Максимальноеколичествовраговнасцене*/
private final int SLIME_COUNT = 15;
    private final int BAT_COUNT = 15;
    private final int ORC_COUNT = 15;
    private final int IMP_COUNT = 15;
    private final int MINOTAUR_COUNT = 15;
    private final int SKELETON_COUNT = 15;
    private final int GHOST_COUNT = 15;
    private final int DARKLORD_COUNT = 5;
    private final int DEATH_COUNT = 1;

public StageScreen() {
    stageActive = false;
    activateTouchpad = true;

    safe = false;
    gameOver = false;
    safeCount = 0;

    battleMusic = Gdx.audio.newMusic ( Gdx.files.internal ( "Audio/bgm/Battle1.ogg" ) );
    battleMusic.setLooping ( true );
    bossBattleMusic = Gdx.audio.newMusic ( Gdx.files.internal ( "Audio/bgm/Battle3.ogg" ) );
    bossBattleMusic.setLooping ( true );
    safeZoneMusic = Gdx.audio.newMusic ( Gdx.files.internal ( "Audio/bgm/Dungeon1.ogg" ) );
    safeZoneMusic.setLooping ( true );
    finishMusic = Gdx.audio.newMusic ( Gdx.files.internal ( "Audio/bgm/Theme1.ogg" ) );
    finishMusic.setLooping ( true );

    //camera
    camera.setToOrtho ( false, Siege.WIDTH, Siege.HEIGHT );
    //camera.setToOrtho ( false, Gdx.graphics.getWidth (), Gdx.graphics.getHeight () );

    midle = new Vector2 ( Siege.WIDTH/2, Siege.HEIGHT/2 );// длятестов
    dispos = new Vector2 ( - 1000, - 1000 );

    texSafeZone = new Texture ( Gdx.files.internal ( "Backgrounds/SafeZone.png" ) );
    texSlimeDungeon = new Texture ( Gdx.files.internal ( "Backgrounds/SlimeDungeon.png" ) );
    texBatDungeon = new Texture ( Gdx.files.internal ( "Backgrounds/BatDungeon.png" ) );
    texOrcDungeon = new Texture ( Gdx.files.internal ( "Backgrounds/OrcDungeon.png" ) );
    texImpDungeon = new Texture ( Gdx.files.internal ( "Backgrounds/ImpDungeon.png" ) );
    texMinotaurDungeon = new Texture ( Gdx.files.internal ( "Backgrounds/MinotaurDungeon.png" )
    );
    texSkeletonDungeon = new Texture ( Gdx.files.internal ( "Backgrounds/SkeletonDungeon.png" ) );
    texGhostDungeon = new Texture ( Gdx.files.internal ( "Backgrounds/GhostDungeon.png" ) );
    texDarkLordDungeon = new Texture ( Gdx.files.internal ( "Backgrounds/DarkLordDungeon.png" )
    );
    texDeathDungeon = new Texture ( Gdx.files.internal ( "Backgrounds/DeathDungeon.png" ) );
    texFinish = new Texture ( Gdx.files.internal ( "Backgrounds/Finish.png" ) );

```

```

texGameOver = new Texture ( Gdx.files.internal ( "Backgrounds/GameOver.png" ) );

//для сенсорного управления
batch = new SpriteBatch ( );

/**Для управления персонажем*/
//create a touchpad skin
playerTouchpadSkin = new Skin ();
//set background image
playerTouchpadSkin.add ( "touchBackground", new Texture ( Gdx.files.internal (
"ActionButtons/touchBackground.png" ) ) );
//set knob image
playerTouchpadSkin.add ( "touchKnob", new Texture ( Gdx.files.internal (
"ActionButtons/touchKnob.png" ) ) );
//create touchpad Style
playerTouchpadStyle = new TouchpadStyle ( );
//Create Drawable's from TouchPad skin
playerTouchBackground = playerTouchpadSkin.getDrawable ( "touchBackground" );
playerTouchKnob = playerTouchpadSkin.getDrawable ( "touchKnob" );
//Apply the Drawables to the TouchPad Style
playerTouchpadStyle.background = playerTouchBackground;
playerTouchpadStyle.knob = playerTouchKnob;
//Create new TouchPad with the created style
playerTouchpad = new Touchpad ( 10, playerTouchpadStyle );
//setBounds(x,y,width,height)
playerTouchpad.setBounds ( 0, 0, 200, 200 );

/**Для управления огнём*/
//create a touchpad skin
fireTouchpadSkin = new Skin ();
//set background image
fireTouchpadSkin.add ( "touchBackground", new Texture ( Gdx.files.internal (
"ActionButtons/touchBackground.png" ) ) );
//set knob image
fireTouchpadSkin.add ( "touchKnob", new Texture ( Gdx.files.internal (
"ActionButtons/touchKnob.png" ) ) );
//create touchpad Style
fireTouchpadStyle = new TouchpadStyle ( );
//Create Drawable's from TouchPad skin
fireTouchBackground = fireTouchpadSkin.getDrawable ( "touchBackground" );
fireTouchKnob = fireTouchpadSkin.getDrawable ( "touchKnob" );
//Apply the Drawables to the TouchPad Style
fireTouchpadStyle.background = fireTouchBackground;
fireTouchpadStyle.knob = fireTouchKnob;
//Create new TouchPad with the created style
fireTouchpad = new Touchpad ( 10, fireTouchpadStyle );
//setBounds(x,y,width,height)
fireTouchpad.setBounds ( Gdx.graphics.getWidth () - 200, 0, 200, 200 );

//Create a Stage and add TouchPad
stageScreenStage = new Stage ( new ScreenViewport ( ) );

if(activateTouchpad) {

```



```

        stageScreenStage.addActor ( playerTouchpad );
        stageScreenStage.addActor ( fireTouchpad );
    }

    //Данные на экране
    texHud = new Texture ( Gdx.files.internal ( "sprites.Player/HUD1.png" ) );
    playerHP = new BitmapFont ();
    score = new BitmapFont ();
    level = new BitmapFont ();
    enemiesDied = new BitmapFont ();

    scoreCount = 0;
    levelCount = 1;
    enemiesDiedCount = 0;

    fireRate = 20;

    fireballstart = new float[ FIREBALL_COUNTS ];
    /**Герой*/
    player = new Player ();
    fireBalls = new FireBall[ FIREBALL_COUNTS ];
    for ( int i = 0; i < FIREBALL_COUNTS; i++ ) {
        fireBalls[ i ] = new FireBall ();
    }
    /**Враги*/
    if ( !safe ) {
        slimes = new EnemySlime[ SLIME_COUNT ];
        for ( int i = 0; i < SLIME_COUNT; i++ ) {
            slimes[ i ] = new EnemySlime ();
        }
    }
}

@Override
protected void handleInput () {
    fireFireBalls ();

    playerMove ();
}

/**Пререндер**)*/

/**Для отрисовки перемещения Героя*/
private void renderPlayer(SpriteBatch batch){

    if(!Gdx.input.isKeyPressed ( Input.Keys.S )&&!Gdx.input.isKeyPressed ( Input.Keys.W
    )&&!Gdx.input.isKeyPressed ( Input.Keys.A )&&!Gdx.input.isKeyPressed ( Input.Keys.D )){
        player.playerDoNothing (batch);
    }

    if (Gdx.input.isKeyPressed (Input.Keys.S)){
        player.renderPlayerDown ( batch );
    }else if (Gdx.input.isKeyPressed (Input.Keys.W)) {

```

```

        player.renderPlayerUp ( batch );
    }
    if (Gdx.input.isKeyPressed (Input.Keys.A)){
        player.renderPlayerLeft ( batch );
    }else if (Gdx.input.isKeyPressed (Input.Keys.D)){
        player.renderPlayerRight ( batch );
    }

    if(activateTouchpad) {
        if ( playerTouchpad.getKnobX () > 140 ) {
            player.renderPlayerRight ( batch );
        }
        if ( playerTouchpad.getKnobX () < 60 ) {
            player.renderPlayerLeft ( batch );
        }
        if ( playerTouchpad.getKnobY () > 140 ) {
            player.renderPlayerUp ( batch );
        }
        if ( playerTouchpad.getKnobY () < 60 ) {
            player.renderPlayerDown ( batch );
        }
    }
}

private void renderSlime(SpriteBatch batch){
/**ОтрисовкаанимациидляперемещенияГероя*/

for ( int i = 0; i < SLIME_COUNT; i++ ) {
    //СправаотГероя
    if(slimes[i].getPositionOfEnemies ().x > player.getPositionOfPlayer ().x
    &&slimes[i].getPositionOfEnemies ().y == player.getPositionOfPlayer ().y){
        slimes[i].renderEnemiesLeft ( batch );
    }
    //СправавнизуотГероя
    if(slimes[i].getPositionOfEnemies ().x > player.getPositionOfPlayer ().x
    &&slimes[i].getPositionOfEnemies ().y < player.getPositionOfPlayer ().y){
        slimes[i].renderEnemiesLeft ( batch );
    }
    //ВнизуотГероя
    if(slimes[i].getPositionOfEnemies ().x == player.getPositionOfPlayer ().x
    &&slimes[i].getPositionOfEnemies ().y < player.getPositionOfPlayer ().y){
        slimes[i].renderEnemiesUp ( batch );
    }
    //СлевавнизуотГероя
    if(slimes[i].getPositionOfEnemies ().x < player.getPositionOfPlayer ().x
    &&slimes[i].getPositionOfEnemies ().y < player.getPositionOfPlayer ().y){
        slimes[i].renderEnemiesRight ( batch );
    }
    //СлеваотГероя
    if(slimes[i].getPositionOfEnemies ().x < player.getPositionOfPlayer ().x
    &&slimes[i].getPositionOfEnemies ().y == player.getPositionOfPlayer ().y){
        slimes[i].renderEnemiesRight ( batch );
    }
    //СлевавверхуотГероя

```

```

        if(slimes[i].getPositionOfEnemies ().x < player.getPositionOfPlayer ().x
        &&slimes[i].getPositionOfEnemies ().y > player.getPositionOfPlayer ().y){
        slimes[i].renderEnemiesRight ( batch );
        }
        //ВверхуотГероя
        if(slimes[i].getPositionOfEnemies ().x == player.getPositionOfPlayer ().x
        &&slimes[i].getPositionOfEnemies ().y > player.getPositionOfPlayer ().y){
        slimes[i].renderEnemiesDown ( batch );
        }
        //СправавверхуотГероя
        if(slimes[i].getPositionOfEnemies ().x > player.getPositionOfPlayer ().x
        &&slimes[i].getPositionOfEnemies ().y > player.getPositionOfPlayer ().y){
        slimes[i].renderEnemiesLeft ( batch );
        }
    }
}
*/
/**Логика*/
private void slimeMoveToPlayer() {
    for ( int i = 0; i < SLIME_COUNT; i++ ) {
        //СправаотГероя
        if ( slimes[ i ].getPositionOfEnemies ().x > player.getPositionOfPlayer ().x &&slimes[ i
        ].getPositionOfEnemies ().y == player.getPositionOfPlayer ().y ) {
        slimes[ i ].getPositionOfEnemies ().x -= slimes[ i ].getSpeed ();
        }
        //СправавнизуотГероя
        if ( slimes[ i ].getPositionOfEnemies ().x > player.getPositionOfPlayer ().x &&slimes[ i
        ].getPositionOfEnemies ().y < player.getPositionOfPlayer ().y ) {
        slimes[ i ].getPositionOfEnemies ().x -= slimes[ i ].getSpeed ();
        slimes[ i ].getPositionOfEnemies ().y += slimes[ i ].getSpeed ();
        }
        //ВнизуотГероя
        if ( slimes[ i ].getPositionOfEnemies ().x == player.getPositionOfPlayer ().x &&slimes[ i
        ].getPositionOfEnemies ().y < player.getPositionOfPlayer ().y ) {
        slimes[ i ].getPositionOfEnemies ().y += slimes[ i ].getSpeed ();
        }
        //СлевавнизуотГероя
        if ( slimes[ i ].getPositionOfEnemies ().x < player.getPositionOfPlayer ().x &&slimes[ i
        ].getPositionOfEnemies ().y < player.getPositionOfPlayer ().y ) {
        slimes[ i ].getPositionOfEnemies ().x += slimes[ i ].getSpeed ();
        slimes[ i ].getPositionOfEnemies ().y += slimes[ i ].getSpeed ();
        }
        //СлеваотГероя
        if ( slimes[ i ].getPositionOfEnemies ().x < player.getPositionOfPlayer ().x &&slimes[ i
        ].getPositionOfEnemies ().y == player.getPositionOfPlayer ().y ) {
        slimes[ i ].getPositionOfEnemies ().x += slimes[ i ].getSpeed ();
        }
        //СлевавверхуотГероя
        if ( slimes[ i ].getPositionOfEnemies ().x < player.getPositionOfPlayer ().x &&slimes[ i
        ].getPositionOfEnemies ().y > player.getPositionOfPlayer ().y ) {
        slimes[ i ].getPositionOfEnemies ().x += slimes[ i ].getSpeed ();
        slimes[ i ].getPositionOfEnemies ().y -= slimes[ i ].getSpeed ();
        }
        //ВверхуотГероя

```

```

        if ( slimes[ i ].getPositionOfEnemies ().x == player.getPositionOfPlayer ().x &&slimes[ i
].getPositionOfEnemies ().y > player.getPositionOfPlayer ().y ) {
slimes[ i ].getPositionOfEnemies ().y -= slimes[ i ].getSpeed ();
        }
        //СправаВверхотГероя
        if ( slimes[ i ].getPositionOfEnemies ().x > player.getPositionOfPlayer ().x &&slimes[ i
].getPositionOfEnemies ().y > player.getPositionOfPlayer ().y ) {
slimes[ i ].getPositionOfEnemies ().x -= slimes[ i ].getSpeed ();
slimes[ i ].getPositionOfEnemies ().y -= slimes[ i ].getSpeed ();
        }
    }
}
private void playerMove(){
    System.out.println ("knob x: " + playerTouchpad.getKnobX ());

    if(activateTouchpad) {
        if ( playerTouchpad.getKnobX () > 140 ) {
            player.right ();
        }
        if ( playerTouchpad.getKnobX () < 60 ) {
            player.left ();
        }
        if ( playerTouchpad.getKnobY () > 140 ) {
            player.up ();
        }
        if ( playerTouchpad.getKnobY () < 60 ) {
            player.down ();
        }
    }
    if (Gdx.input.isKeyPressed (Input.Keys.S)){
        player.down ();
    }
    if (Gdx.input.isKeyPressed (Input.Keys.A)){
        player.left ();
    }
    if (Gdx.input.isKeyPressed (Input.Keys.D)){
        player.right ();
    }
    if (Gdx.input.isKeyPressed (Input.Keys.W)){
        player.up ();
    }
}

/**ДлястрельбыОгненнымишарами*/
private void fireFireBalls(){

    if(activateTouchpad) {
        if ( fireTouchpad.getKnobX () > 140 ) {
            fireCounterRight++;
            if ( fireCounterRight > fireRate ) {
                fireCounterRight = 0;
                fireballActivate ( 0, 0, 10 );
            }
        }
    }
}

```

```

if ( fireTouchpad.getKnobX () < 60 ) {
    fireCounterLeft++;
    if ( fireCounterLeft > fireRate ) {
        fireCounterLeft = 0;
        fireballActivate ( 0, 0, 10 );
    }
}
if ( fireTouchpad.getKnobY () > 140 ) {
    fireCounterUp++;
    if ( fireCounterUp > fireRate ) {
        fireCounterUp = 0;
        fireballActivate ( 0, 0, 10 );
    }
}
if ( fireTouchpad.getKnobY () < 60 ) {
    fireCounterDown++;
    if ( fireCounterDown > fireRate ) {
        fireCounterDown = 0;
        fireballActivate ( 0, 0, 10 );
    }
}
}
if( Gdx.input.isKeyPressed ( Input.Keys.RIGHT )){
    fireCounterRight++;
    if(fireCounterRight > fireRate){
        fireCounterRight = 0;
        fireballActivate ( 0, 0, 10 );
    }
}
else {
    if ( Gdx.input.isKeyPressed ( Input.Keys.LEFT ) ) {
        fireCounterLeft++;
        if(fireCounterLeft > fireRate){
            fireCounterLeft = 0;
            fireballActivate ( 0, 0, 10 );
        }
    }
}
if( Gdx.input.isKeyPressed ( Input.Keys.UP )){
    fireCounterUp++;
    if(fireCounterUp > fireRate){
        fireCounterUp = 0;
        fireballActivate ( 0, 0, 10 );
    }
}
else {
    if ( Gdx.input.isKeyPressed ( Input.Keys.DOWN ) ) {
        fireCounterDown++;
        if(fireCounterDown > fireRate){
            fireCounterDown = 0;
            fireballActivate ( 0, 0, 10 );
        }
    }
}
}
}

```

```

private void fireballActivate(float dx, float dy, float speed){

    for ( int i = 0; i < FIREBALL_COUNTS; i++ ) {
        if(!fireBalls[i].isActive ()){
fireBalls[i].getFireSound ().play ();
fireBalls[i].setup ( player.getPositionOfPlayer ().x + dx, player.getPositionOfPlayer ().y + dy, speed
);
            break;
        }
    }
}

private void fireballupdate(){

    if(activateTouchpad) {
        if ( fireTouchpad.getKnobX () > 140 ) {
            for ( int i = 0; i < FIREBALL_COUNTS; i++ ) {
fireBalls[ i ].fireRight ();
            }
        }
        if ( fireTouchpad.getKnobX () < 60 ) {
            for ( int i = 0; i < FIREBALL_COUNTS; i++ ) {
fireBalls[ i ].fireLeft ();
            }
        }
        if ( fireTouchpad.getKnobY () > 140 ) {
            for ( int i = 0; i < FIREBALL_COUNTS; i++ ) {
fireBalls[ i ].fireUp ();
            }
        }
        if ( fireTouchpad.getKnobY () < 60 ) {
            for ( int i = 0; i < FIREBALL_COUNTS; i++ ) {
fireBalls[ i ].fireDown ();
            }
        }
    }

    if( Gdx.input.isKeyPressed ( Input.Keys.RIGHT )){
        //System.out.println ("RIGHT");
        for ( int i = 0; i < FIREBALL_COUNTS; i++ ) {
fireBalls[i].fireRight ();
        }
    }else{
        if(Gdx.input.isKeyPressed ( Input.Keys.LEFT )){
            for ( int i = 0; i < FIREBALL_COUNTS; i++ ) {
fireBalls[i].fireLeft ();
            }
        }
    }

    if( Gdx.input.isKeyPressed ( Input.Keys.UP )){
        for ( int i = 0; i < FIREBALL_COUNTS; i++ ) {
fireBalls[i].fireUp ();
        }
    }else{
        if(Gdx.input.isKeyPressed ( Input.Keys.DOWN )){

```

```

        for ( int i = 0; i < FIREBALL_COUNTS; i++ ) {
fireBalls[i].fireDown ();
        }
    }
}

/** Конец стрельбы огненными шарами */

/** Коллизия врагов с персонажем */
private void playerCollision() {

    if ( !safe ) {
        for ( int i = 0; i < SLIME_COUNT; i++ ) {
            if ( player.getRectOfPlayer ().overlaps ( slimes[ i ].getRectOfEnemies () ) ) {
                player.getTakeDamage ().play ();
                player.getDamage ( slimes[ i ].getDamage () );
                player.teleport ();
                break;
            }
        }
    }
}

/** Коллизия врагов с огненными шарами */
private void fireballsCollision() {

    if ( !safe ) {
        if ( levelCount == 1 ) {
            for ( int i = 0; i < SLIME_COUNT; i++ ) {
                for ( int j = 0; j < FIREBALL_COUNTS; j++ ) {
                    if ( fireBalls[ j ].getRectangleOfFireball ().overlaps ( slimes[ i ].getRectOfEnemies ()
                ) ) {
slimes[ i ].getDamage ( player.getDamage () );
                    if ( slimes[ i ].isRecreate () ) {
                        enemiesDiedCount++;
                        scoreCount += slimes[ i ].getScore ();
                    }
                    fireBalls[ j ].destroy ();
                    break;
                }
            }
        }
    }
}

private void fireballsNoAtak(){
    for ( int i = 0; i < FIREBALL_COUNTS; i++ ) {
        if(fireBalls[i].isActive ()){
            fireballstart[i] += 1;
            if(fireballstart[i] > 150){/**100*/
                fireBalls[i].destroy ();
                fireballstart[i] = 0;
            }
        }
    }
}

```

```

    }
}

/**Переключениемеждууровнями*/
private void levels(){

    if(enemiesDiedCount == LEVEL2 && safeCount == 0){
        safeCount = 1;
    }
    if(safeCount == 1){
        for ( int i = 0; i < SLIME_COUNT; i++ ) {
            slimes[i].setPositionOfEnemies ( dispos );
        }
        player.setHp ( 100 );
        safe = true;
        if(player.getPositionOfPlayer ().x > Siege.WIDTH/2 - 48 && player.getPositionOfPlayer
().x < (Siege.WIDTH/2 ) &&
            player.getPositionOfPlayer ().y > Siege.HEIGHT - 96) {
            player.setPositionOfPlayer ( midle );
            safeCount = 2;
            safe = false;
            levelCount = 2;
        }
    }

    if(enemiesDiedCount == LEVEL3 && safeCount == 2){
        safeCount = 3;
    }
    if(safeCount == 3){
        for ( int i = 0; i < BAT_COUNT; i++ ) {
            bats[i].setPositionOfEnemies ( dispos );
        }
        player.setHp ( 100 );
        safe = true;
        if(player.getPositionOfPlayer ().x > Siege.WIDTH/2 - 48 && player.getPositionOfPlayer
().x < (Siege.WIDTH/2 ) &&
            player.getPositionOfPlayer ().y > Siege.HEIGHT - 96){
            player.setPositionOfPlayer ( midle );
            safeCount = 4;
            safe = false;
            levelCount = 3;
        }
    }

    if(enemiesDiedCount == LEVEL4 && safeCount == 4){
        safeCount = 5;
    }
    if(safeCount == 5){
        for ( int i = 0; i < ORC_COUNT; i++ ) {
            orcs[i].setPositionOfEnemies ( dispos );
        }
        player.setHp ( 100 );
        safe = true;
    }
}

```



```

        if(player.getPositionOfPlayer ().x > Siege.WIDTH/2 - 48 && player.getPositionOfPlayer
().x < (Siege.WIDTH/2 ) &&
            player.getPositionOfPlayer ().y > Siege.HEIGHT - 96){
            player.setPositionOfPlayer ( midle );
            safeCount = 6;
            safe = false;
            levelCount = 4;
        }
    }

    if(enemiesDiedCount == LEVEL5 && safeCount == 6){
        safeCount = 7;
    }
    if(safeCount == 7){
        for ( int i = 0; i < IMP_COUNT; i++ ) {
imps[i].setPositionOfEnemies ( dispos );
        }
        player.setHp ( 100 );
        safe = true;
        if(player.getPositionOfPlayer ().x > Siege.WIDTH/2 - 48 && player.getPositionOfPlayer
().x < (Siege.WIDTH/2 ) &&
            player.getPositionOfPlayer ().y > Siege.HEIGHT - 96){
            player.setPositionOfPlayer ( midle );
            safeCount = 8;
            safe = false;
            levelCount = 5;
        }
    }

    if(enemiesDiedCount == LEVEL6 && safeCount == 8){
        safeCount = 9;
    }
    if(safeCount == 9){
        for ( int i = 0; i < MINOTAUR_COUNT; i++ ) {
minotaurs[i].setPositionOfEnemies ( dispos );
        }
        player.setHp ( 100 );
        safe = true;
        if(player.getPositionOfPlayer ().x > Siege.WIDTH/2 - 48 && player.getPositionOfPlayer
().x < (Siege.WIDTH/2 ) &&
            player.getPositionOfPlayer ().y > Siege.HEIGHT - 96){
            player.setPositionOfPlayer ( midle );
            safeCount = 10;
            safe = false;
            levelCount = 6;
        }
    }

    if(enemiesDiedCount == LEVEL7 && safeCount == 10){
        safeCount = 11;
    }
    if(safeCount == 11){
        for ( int i = 0; i < SKELETON_COUNT; i++ ) {
skeletons[i].setPositionOfEnemies ( dispos );
    }

```

```

    }
    player.setHp ( 100 );
    safe = true;
    if(player.getPositionOfPlayer ().x > Siege.WIDTH/2 - 48 && player.getPositionOfPlayer
().x < (Siege.WIDTH/2 ) &&
        player.getPositionOfPlayer ().y > Siege.HEIGHT - 96){
        player.setPositionOfPlayer ( midle );
        safeCount = 12;
        safe = false;
        levelCount = 7;
    }
}

if(enemiesDiedCount == LEVEL8 && safeCount == 12){
    safeCount = 13;
}
if(safeCount == 13){
    for ( int i = 0; i < GHOST_COUNT; i++ ) {
ghosts[i].setPositionOfEnemies ( dispos );
    }
    player.setHp ( 100 );
    safe = true;
    if(player.getPositionOfPlayer ().x > Siege.WIDTH/2 - 48 && player.getPositionOfPlayer
().x < (Siege.WIDTH/2 ) &&
        player.getPositionOfPlayer ().y > Siege.HEIGHT - 96){
        player.setPositionOfPlayer ( midle );
        safeCount = 14;
        safe = false;
        levelCount = 8;
    }
}

if(enemiesDiedCount == LEVEL9 && safeCount == 14){
    safeCount = 15;
}
if(safeCount == 15){
    for ( int i = 0; i < DARKLORD_COUNT; i++ ) {
darkLords[i].setPositionOfEnemies ( dispos );
    }
    player.setHp ( 100 );
    safe = true;
    if(player.getPositionOfPlayer ().x > Siege.WIDTH/2 - 48 && player.getPositionOfPlayer
().x < (Siege.WIDTH/2 ) &&
        player.getPositionOfPlayer ().y > Siege.HEIGHT - 96){
        player.setPositionOfPlayer ( midle );
        safeCount = 16;
        safe = false;
        levelCount = 9;
    }
}

if(enemiesDiedCount == FINISH && safeCount == 16){
    safeCount = 17;
}

```

```

        if(enemiesDiedCount == FINISH && safeCount == 17){
            for ( int i = 0; i < DEATH_COUNT; i++ ) {
deaths[i].setPositionOfEnemies ( dispos );
            }
            levelCount = 10;
            player.setHp ( 100 );
            safe = true;
            System.out.println ("You win");
            if(player.getPositionOfPlayer ().x > Siege.WIDTH/2 - 48 && player.getPositionOfPlayer
().x < (Siege.WIDTH/2 ) &&
                player.getPositionOfPlayer ().y > Siege.HEIGHT - 96){
                Gdx.app.exit ();
            }
        }
    }

private void appout(){
    if(player.getHp () <= 0){
        gameOver = true;
        System.out.println ("You Dead");
        if(Gdx.input.isTouched ()) {
            Gdx.app.exit ();
        }
    }
}

public void render(SpriteBatch batch){

    batch.setProjectionMatrix ( camera.combined );
    camera.update ();

    if(stageActive) {
        if(safeCount == 1 || safeCount == 3 || safeCount == 5 || safeCount == 7 || safeCount == 9 ||
safeCount == 11 || safeCount == 13 || safeCount == 15){
            batch.draw ( texSafeZone, 0, 0, Siege.WIDTH, Siege.HEIGHT );
        }
        if ( levelCount == 1 && safeCount == 0) {
            batch.draw ( texSlimeDungeon, 0, 0, Siege.WIDTH, Siege.HEIGHT );
        }
        if ( levelCount == 2 && safeCount == 2) {
            batch.draw ( texBatDungeon, 0, 0, Siege.WIDTH, Siege.HEIGHT );
        }
        if ( levelCount == 3 && safeCount == 4) {
            batch.draw ( texOrcDungeon, 0, 0, Siege.WIDTH, Siege.HEIGHT );
        }
        if ( levelCount == 4 && safeCount == 6) {
            batch.draw ( texImpDungeon, 0, 0, Siege.WIDTH, Siege.HEIGHT );
        }
        if ( levelCount == 5 && safeCount == 8) {
            batch.draw ( texMinotaurDungeon, 0, 0, Siege.WIDTH, Siege.HEIGHT );
        }
        if ( levelCount == 6 && safeCount == 10) {
            batch.draw ( texSkeletonDungeon, 0, 0, Siege.WIDTH, Siege.HEIGHT );
        }
    }
}

```

```

if ( levelCount == 7 && safeCount == 12) {
    batch.draw ( texGhostDungeon, 0, 0, Siege.WIDTH, Siege.HEIGHT );
}
if ( levelCount == 8 && safeCount == 14) {
    batch.draw ( texDarkLordDungeon, 0, 0, Siege.WIDTH, Siege.HEIGHT );
}
if ( levelCount == 9 && safeCount == 16) {
    batch.draw ( texDeathDungeon, 0, 0, Siege.WIDTH, Siege.HEIGHT );
}
if ( levelCount == 10 && safeCount == 17) {
    batch.draw ( texFinish, 0, 0, Siege.WIDTH, Siege.HEIGHT );
}

renderPlayer (batch);

for ( int i = 0; i < FIREBALL_COUNTS; i++ ) {
    if ( fireBalls[ i ].isActive () ) {
fireBalls[ i ].fireBallAnimation ( batch );
    }
}

if ( levelCount == 1 && !safe) {
    renderSlime ( batch );
}
if ( levelCount == 2 && !safe) {
    renderBat ( batch );
}
if ( levelCount == 3 && !safe) {
    renderOrc ( batch );
}
if ( levelCount == 4 && !safe) {
    renderImp ( batch );
}
if ( levelCount == 5 && !safe) {
    renderMinotaur ( batch );
}
if ( levelCount == 6 && !safe) {
    renderSkeleton ( batch );
}
if ( levelCount == 7 && !safe) {
    renderGhost ( batch );
}
if ( levelCount == 8 && !safe) {
    renderDarkLord ( batch );
}
if ( levelCount == 9 && !safe) {
    renderDeath ( batch );
}

batch.draw (texHud, Siege.WIDTH/2 - texHud.getWidth ()/2, 0);
playerHP.draw ( batch, "Hit Points: " + player.getHp (), Siege.WIDTH/2, 144 );
score.draw ( batch, "Score: " + scoreCount, Siege.WIDTH/2, 124 );
level.draw ( batch, "Level: " + levelCount, Siege.WIDTH/2, 104 );

```

```

        enemiesDied.draw ( batch, "Enemies died: " + enemiesDiedCount, Siege.WIDTH/2, 84 );

        if(gameOver){
            batch.draw ( texGameOver, 0, 0, Siege.WIDTH, Siege.HEIGHT );
        }

        /**отрисовывающёраз HUD, таккакбезэтого не отображается enemiesDied.draw*/
        //playerHP.draw ( batch, "asdadsad",0,0 );
        //stageScreenStage.draw ();
    }
}

public void renderStage(){
    if(activateTouchpad) {
        stageScreenStage.draw ();
    }
}

public void update(){
    if(stageActive) {

        if(!gameOver) {
            if ( Gdx.input.justTouched () ) {
                System.out.println ( "lvl " + levelCount );
                System.out.println ( "Width " + Gdx.graphics.getWidth () );
            }

            //tests ();

            levels ();

            player.update ();

            handleInput ();

            fireballupdate ();

            /**Логикадвиженияврагов*/
            if ( levelCount == 1 && ! safe ) {
                slimeMoveToPlayer ();
            }
            for ( int i = 0; i < SLIME_COUNT; i++ ) {
                slimes[ i ].update ();
            }

            /**Проверкаколлизии*/
            playerCollision ();
            fireballsCollision ();

            /**"Время" нахожденияогненныхшаровнаэкране*/
            fireballsNoAtak ();
        }
        appout ();
    }
}

```

```

    }

    public void dispose(){
        //batch.dispose ();
        battleMusic.dispose ();
        bossBattleMusic.dispose ();
        safeZoneMusic.dispose ();
        finishMusic.dispose ();
        texHud.dispose ();
        texSlimeDungeon.dispose ();
        texBatDungeon.dispose ();
        texOrcDungeon.dispose ();
        texImpDungeon.dispose ();
        texMinotaurDungeon.dispose ();
        texSkeletonDungeon.dispose ();
        texGhostDungeon.dispose ();
        texDarkLordDungeon.dispose ();
        texDeathDungeon.dispose ();
        texSafeZone.dispose ();
        for ( int i = 0; i < FIREBALL_COUNTS; i++ ) {
            fireBalls[i].dispose ();
        }
        for ( int i = 0; i < SLIME_COUNT; i++ ) {
            slimes[i].dispose ();
        }
        for ( int i = 0; i < BAT_COUNT; i++ ) {
            bats[i].dispose ();
        }
        for ( int i = 0; i < ORC_COUNT; i++ ) {
            orcs[i].dispose ();
        }
        for ( int i = 0; i < IMP_COUNT; i++ ) {
            imps[i].dispose ();
        }
        for ( int i = 0; i < MINOTAUR_COUNT; i++ ) {
            minotaurs[i].dispose ();
        }
        for ( int i = 0; i < SKELETON_COUNT; i++ ) {
            skeletons[i].dispose ();
        }
        for ( int i = 0; i < GHOST_COUNT; i++ ) {
            ghosts[i].dispose ();
        }
        for ( int i = 0; i < DARKLORD_COUNT; i++ ) {
            darkLords[i].dispose ();
        }
        for ( int i = 0; i < DEATH_COUNT; i++ ) {
            deaths[i].dispose ();
        }
        playerHP.dispose ();
        score.dispose ();
        level.dispose ();
        enemiesDied.dispose ();
    }

```


Додаток Е. Лістинг коду головного героя Player.java

```
package com.crossgame.siege.Hero;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.Input;
import com.badlogic.gdx.audio.Sound;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.Animation;
import com.badlogic.gdx.graphics.g2d.SpriteBatch;
import com.badlogic.gdx.graphics.g2d.TextureRegion;
import com.badlogic.gdx.math.Rectangle;
import com.badlogic.gdx.math.Vector2;
import com.crossgame.siege.Siege;

/**
 * Created by Fenriron 22.01.2017.
 */

public class Player {

    private static final int FRAME_COLS = 3; //Ширина атласа
    private static final int FRAME_ROWS = 1; //Длина атласа

    //Для анимаций перемещения
    private Animation playerWalkAnimationDown;
    private Animation playerWalkAnimationLeft;
    private Animation playerWalkAnimationRight;
    private Animation playerWalkAnimationUp;
    private Animation playerTeleportAnimation;

    //Для атласов
    private Texture playerWalkDown;
    private Texture playerWalkLeft;
    private Texture playerWalkRight;
    private Texture playerWalkUp;
    private Texture playerTeleport;

    //Для бездействия Героя
    private TextureRegion playerStayDown;
    private TextureRegion playerStayLeft;
    private TextureRegion playerStayRight;
    private TextureRegion playerStayUp;

    //Массивы для хранения кадров анимации
    private TextureRegion[] playerWalkFramesDown;
    private TextureRegion[] playerWalkFramesLeft;
    private TextureRegion[] playerWalkFramesRight;
    private TextureRegion[] playerWalkFramesUp;
    private TextureRegion[] playerTeleportFrames;

    //Для выделения кадра из атласа
    private TextureRegion playerCurrentFrameDown;
    private TextureRegion playerCurrentFrameLeft;
```



```
private TextureRegion playerCurrentFrameRight;
private TextureRegion playerCurrentFrameUp;
private TextureRegion playerCurrentFrameTeleport;
```

```
//Переменныедляанимации
```

```
private float playerStateTimeDown;
private float playerStateTimeLeft;
private float playerStateTimeRight;
private float playerStateTimeUp;
private float playerStateTimeTeleport;
```

```
private Sound takeDamage;
public Sound getTakeDamage(){
    return takeDamage;
}
private Sound heal;
public Sound getHeal(){
    return heal;
}
```

```
private boolean DownisActive;
private boolean LeftisActive;
private boolean RightisActive;
private boolean UpisActive;
```

```
private SpriteBatch batch;
```

```
private Vector2 positionOfPlayer;//ПозицияГероя
```

```
private Rectangle rectOfPlayer;//ХитбоксГероя
```

```
//Геттеры
```

```
public Rectangle getRectOfPlayer(){
    return rectOfPlayer;
}
```

```
public Vector2 getPositionOfPlayer(){
    return positionOfPlayer;
}
```

```
public int getHp(){
    return hp;
}
```

```
public int getDamage(){
    return damage;
}
```

```
//Сеттеры
```

```
public Vector2 setPositionOfPlayer(Vector2 pos){
    this.positionOfPlayer = pos;
    return positionOfPlayer;
}
```

```

public void setHp ( int hp ) {
    this.hp = hp;
}

//ХарактеристикаГероя
private float speed;//СкоростьперемещенияГероя
private int hp;//HitPoints of Player
private int damage;//УронГероя
//private int fireCounterRight;
//private int fireCounterLeft;
//private int fireCounterUp;
//private int fireCounterDown;
//private int fireRate;

public Player(){
    //Задаёмскоростьперсонажа
    speed = 4.0f;
    hp = 100;
    damage = 2;
    //fireRate = 10;

    //ЗадаёмначальнуюпозициюГероя
    positionOfPlayer = new Vector2 ( Siege.WIDTH/2,Siege.HEIGHT/2 );

    //СоздаёмхитбоксГероя
    rectOfPlayer = new Rectangle ( positionOfPlayer.x, positionOfPlayer.y, 48, 48 );

    //Загружаематласы    PlayerGoDown.png
    playerWalkDown = new Texture ( "sprites.Player/PlayerGoDown.png" );
    playerWalkLeft = new Texture ( "sprites.Player/PlayerGoLeft.png" );
    playerWalkRight = new Texture ( "sprites.Player/PlayerGoRight.png" );
    playerWalkUp = new Texture ( "sprites.Player/PlayerGoUp.png" );
    playerTeleport = new Texture ( "sprites.Player/Teleport.png" );

    takeDamage = Gdx.audio.newSound ( Gdx.files.internal ( "Audio/se/Damage3.ogg" ) );
    heal = Gdx.audio.newSound ( Gdx.files.internal ( "Audio/se/Heal1.ogg" ) );

    //ВырезаемкадрыизатласовдлябездействияГероя
    playerStayDown = new TextureRegion ( playerWalkDown, 48,0, playerWalkDown.getWidth
() / FRAME_COLS, playerWalkDown.getHeight () / FRAME_ROWS );
    //playerStayLeft = new TextureRegion ( playerWalkLeft, 48,0, playerWalkLeft.getWidth
() / FRAME_COLS, playerWalkLeft.getHeight () / FRAME_ROWS );
    //playerStayRight = new TextureRegion ( playerWalkRight, 48,0, playerWalkRight.getWidth
() / FRAME_COLS, playerWalkRight.getHeight () / FRAME_ROWS );
    //playerStayUp = new TextureRegion ( playerWalkUp, 48,0, playerWalkUp.getWidth
() / FRAME_COLS, playerWalkUp.getHeight () / FRAME_ROWS );

    //playerWalkDown    //Дляанимацийперемещениявниз
    TextureRegion[][] playerWD = TextureRegion.split ( playerWalkDown,
playerWalkDown.getWidth() / FRAME_COLS, playerWalkDown.getHeight() / FRAME_ROWS );
    playerWalkFramesDown = new TextureRegion[FRAME_COLS * FRAME_ROWS];
    int indexDown = 0;
    for ( int i = 0; i < FRAME_ROWS; i++ ) {
        for ( int j = 0; j < FRAME_COLS; j++ ) {

```

```

        if(i == 0 ) {
            playerWalkFramesDown[ indexDown++ ] = playerWD[ i ][ j ];
            //System.out.println (index);
        }
    }
}
playerWalkAnimationDown = new Animation ( 1/10f, playerWalkFramesDown );
batch = new SpriteBatch ();
playerStateTimeDown = 0f;

//playerWalkLeft //ДляанимацийперемещенияВлево
TextureRegion[][] playerWL = TextureRegion.split (playerWalkLeft,
playerWalkLeft.getWidth()/FRAME_COLS, playerWalkLeft.getHeight()/FRAME_ROWS);
playerWalkFramesLeft = new TextureRegion[FRAME_COLS * FRAME_ROWS];
int indexLeft = 0;
for ( int i = 0; i <FRAME_ROWS; i++ ) {
    for ( int j = 0; j <FRAME_COLS; j++ ) {
        if(i == 0 ) {
            playerWalkFramesLeft[ indexLeft++ ] = playerWL[ i ][ j ];
            //System.out.println (index);
        }
    }
}
playerWalkAnimationLeft = new Animation ( 1/10f, playerWalkFramesLeft );
batch = new SpriteBatch ();
playerStateTimeLeft = 0f;

//playerWalkRight //ДляанимацийперемещенияВправо
TextureRegion[][] playerWR = TextureRegion.split (playerWalkRight,
playerWalkRight.getWidth()/FRAME_COLS, playerWalkRight.getHeight()/FRAME_ROWS);
playerWalkFramesRight = new TextureRegion[FRAME_COLS * FRAME_ROWS];
int indexRight = 0;
for ( int i = 0; i <FRAME_ROWS; i++ ) {
    for ( int j = 0; j <FRAME_COLS; j++ ) {
        if(i == 0 ) {
            playerWalkFramesRight[ indexRight++ ] = playerWR[ i ][ j ];
            //System.out.println (index);
        }
    }
}
playerWalkAnimationRight = new Animation ( 1/10f, playerWalkFramesRight );
batch = new SpriteBatch ();
playerStateTimeRight = 0f;

//playerWalkUp //ДляанимацийперемещенияВверх
TextureRegion[][] playerWU = TextureRegion.split (playerWalkUp,
playerWalkUp.getWidth()/FRAME_COLS, playerWalkUp.getHeight()/FRAME_ROWS);
playerWalkFramesUp = new TextureRegion[FRAME_COLS * FRAME_ROWS];
int indexUp = 0;
for ( int i = 0; i <FRAME_ROWS; i++ ) {
    for ( int j = 0; j <FRAME_COLS; j++ ) {
        if(i == 0 ) {
            playerWalkFramesUp[ indexUp++ ] = playerWU[ i ][ j ];
            //System.out.println (index);
        }
    }
}

```

```

    }
    }
}
playerWalkAnimationUp = new Animation ( 1/10f, playerWalkFramesUp );
batch = new SpriteBatch ();
playerStateTimeUp = 0f;

//playerTeleport
/*TextureRegion[][] playerTP = TextureRegion.split ( playerTeleport, playerTeleport.getWidth
()/FRAME_COLS, playerTeleport.getHeight ()/FRAME_ROWS );
playerTeleportFrames = new TextureRegion[FRAME_COLS * FRAME_ROWS];
int indexTeleport = 0;
for ( int i = 0; i < FRAME_ROWS; i++ ) {
    for ( int j = 0; j < FRAME_COLS; j++ ) {
        if(i == 0){
            playerTeleportFrames[indexTeleport++] = playerTP[i][j];
        }
    }
}
playerTeleportAnimation = new Animation ( 1/10, playerTeleportFrames );
batch = new SpriteBatch ( );
playerStateTimeTeleport = 0f;*/
}

/*public void teleportAnimation(SpriteBatch batch){
    playerStateTimeTeleport += Gdx.graphics.getDeltaTime ();
    playerCurrentFrameTeleport = ( TextureRegion ) playerTeleportAnimation.getKeyFrame (
playerStateTimeTeleport, false );
    batch.draw ( playerCurrentFrameTeleport, positionOfPlayer.x, positionOfPlayer.y);
}*/

public void teleport(){
    positionOfPlayer = new Vector2 ( (float)Math.random () * Siege.WIDTH, (float)Math.random
() * Siege.HEIGHT);
}

public void getDamage(int dmg){
    hp -= dmg;
}

//ДляорганизациииуправленияГероем

public void down(){
    positionOfPlayer.y -= speed;
    if(positionOfPlayer.y < -48){
        positionOfPlayer.y = Siege.HEIGHT + 48;
    }
}

public void left(){
    positionOfPlayer.x -= speed;
    if(positionOfPlayer.x < -48){
        positionOfPlayer.x = Siege.WIDTH + 48;
    }
}

```

```

    }
}

public void right(){
    positionOfPlayer.x += speed;
    if(positionOfPlayer.x > Siege.WIDTH + 48){
        positionOfPlayer.x = -48;
    }
}

public void up(){
    positionOfPlayer.y += speed;
    if(positionOfPlayer.y >Siege.HEIGHT +48){
        positionOfPlayer.y = -48;
    }
}

private void Input(){
    //down ();
    //left ();
    //right ();
    //up ();
}

public void playerDoNothing(SpriteBatch batch){
    //ДляотрисовкибездействияГероя
    batch.draw ( playerStayDown, positionOfPlayer.x, positionOfPlayer.y );
}

public void renderPlayerDown(SpriteBatch batch){
    playerStateTimeDown += Gdx.graphics.getDeltaTime ();
    playerCurrentFrameDown = ( TextureRegion ) playerWalkAnimationDown.getKeyFrame (
playerStateTimeDown,true );
    //Input ();
    batch.draw ( playerCurrentFrameDown, positionOfPlayer.x, positionOfPlayer.y);
}

public void renderPlayerLeft(SpriteBatch batch){
    playerStateTimeLeft += Gdx.graphics.getDeltaTime ();
    playerCurrentFrameLeft = ( TextureRegion ) playerWalkAnimationLeft.getKeyFrame (
playerStateTimeLeft,true );
    //Input ();
    batch.draw ( playerCurrentFrameLeft, positionOfPlayer.x, positionOfPlayer.y);
}

public void renderPlayerRight(SpriteBatch batch){
    playerStateTimeRight += Gdx.graphics.getDeltaTime ();
    playerCurrentFrameRight = ( TextureRegion ) playerWalkAnimationRight.getKeyFrame (
playerStateTimeRight,true );
    //Input ();
    batch.draw ( playerCurrentFrameRight, positionOfPlayer.x, positionOfPlayer.y);
}

public void renderPlayerUp(SpriteBatch batch){

```

```

        playerStateTimeUp += Gdx.graphics.getDeltaTime ();
        playerCurrentFrameUp = ( TextureRegion ) playerWalkAnimationUp.getKeyFrame (
playerStateTimeUp, true );
        //Input ();
        batch.draw ( playerCurrentFrameUp, positionOfPlayer.x, positionOfPlayer.y );
    }

    public void render(SpriteBatch batch){

        if(!Gdx.input.isKeyPressed ( Input.Keys.S )&&!Gdx.input.isKeyPressed ( Input.Keys.W
)&&!Gdx.input.isKeyPressed ( Input.Keys.A )&&!Gdx.input.isKeyPressed ( Input.Keys.D )){
            playerDoNothing (batch);
        }
    }

    public void update() {
        //Input ();
        rectOfPlayer.x = positionOfPlayer.x;
        rectOfPlayer.y = positionOfPlayer.y;
    }

    public void dispose(){
        takeDamage.dispose ();
        heal.dispose ();
    }
}

```

Міністерство освіти і науки України
Державний заклад «Луганський національний університет
імені Тараса Шевченка»
Факультет (інститут) Навчально-науковий інститут математики та
інформаційних технологій

(повна назва)
Кафедра Інформаційних технологій та систем

(повна назва)

КЕРІВНИЦТВО КОРИСТУВАЧА
на виконання програмної розробки (ПР):
"РОЗРОБКА КРОСПЛАТФОРМНОЇ 2D ГРИ ЗАСОБАМИ JAVA ТА
LIBGDX"
ІТС. ІПЗ4.1224-02-КК

ПОГОДЖЕНО
Керівник кваліфікаційної роботи

Донченко В. В.

“ ” 2024р.

ВИКОНАВЕЦЬ
Студент групи 4 ІПЗ

Вернік О. В.

“ ” 2024р.

Полтава 2024

ЗМІСТ

ВСТУП	3
1. Підготовка до роботи з додатком.....	4
2. Робота з додатком	5

ВСТУП

Кросплатформна графічна гра «Sege» призначена для використання у розважальних цілях.

Графічна гра «Siege» - це самостійна програма, яка має інтуїтивний користувальницький інтерфейс, проста у використанні.

У даній грі ви маєте взяти на себе роль головного героя Піроманта, якому необхідно проходячи рівень за рівнем дістатися до Мармурового палацу. На шляху до палацу його чекають небезпека у вигляді ворогів, які намагатимуться усіма своїми силами зупинити головного героя.

Підготовка до роботи з додатком

Персональні комп'ютери

Кросплатформна графічна гра «Siege» була розроблена мовою програмування Java. Тому для того щоб на ПК було можливо запустити цю гру необхідна наявність програмного забезпечення Java. Якщо на вашому ПК це ПЗ не встановлено, його можливо завантажити з офіційного сайту Oracle Corporation [US] java.com/ru/download/. Для того, щоб його встановити треба запустити завантажений файл та слідувати вказівкам.

Смартфони на базі ОС Android

Єдині технічні умови для цієї платформи – це те, що на смартфонах повина бути встановлена ОС Android версії 4.4. Якщо ви не знаєте версію вашого ОС Android, то його можливо подивитися у налаштуваннях смартфонів у графі «О телефоні». Якщо версія нижче необхідної, то треба подивитися в Інтернеті чи підтримує ваш смартфон дану версію ОС. Якщо так, то завантажити нову версію Android можливо декількома способами. Перший спосіб – це «повітряне завантаження», тобто за допомогою wifi у налаштуваннях смартфона в графі «О телефоні» можливо завантажити нову версію, слідуючи вказівкам телефону. Другий спосіб – це завантажити до смартфона нову версію ОС за допомогою комп'ютера. Для цього є спеціальні програми, які зазвичай йдуть у комплекті з телефоном. Запустіть програму на ПК, потім підключіть телефон до нього через USB кабель. Потім у програмі оберіть «Завантажити нову версію ОС Android». Далі слідувати вказівкам програми.

Робота з додатком

Робота на ПК

Взаємодія між інтерфейсом та користувачем здійснюється за допомогою мишки.

Керування головним героєм проводиться через клавіатуру, а саме через клавіші **WASD**. Може бути, що головний герой не переміщується. Це пов'язано з тим, що розкладка клавіатури не співпадає з ігровими налаштуваннями. Для того щоб це виправити необхідно увімкнути «англійську» розкладку клавіатури.

Керування вогнем проводиться через клавіатуру, а саме клавішами →←↓↑.

Гру, запущену на ПК, зображено на Рис.1

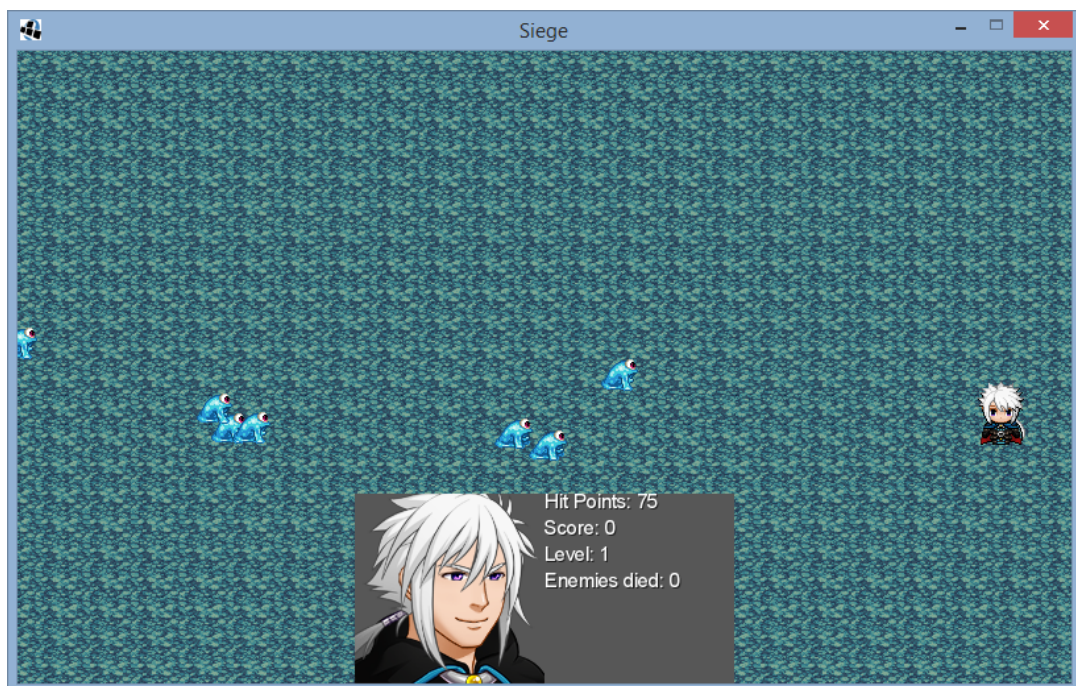


Рис. 1. Гра «Siege» запущена на ПК

Робота на смартфоні

Взаємодія між інтерфейсом та користувачем здійснюється через екран смартфона, тобто через дотик.

Для керування головним героєм та вогнем у грі передбачена можливість увімкнути сенсорні джойстики у головному меню. Лівий джойстик відповідає за керування героєм, правий – за керування вогнем.

Гру, запущену на емуляторі смартфоні, зображено на Рис. 2.

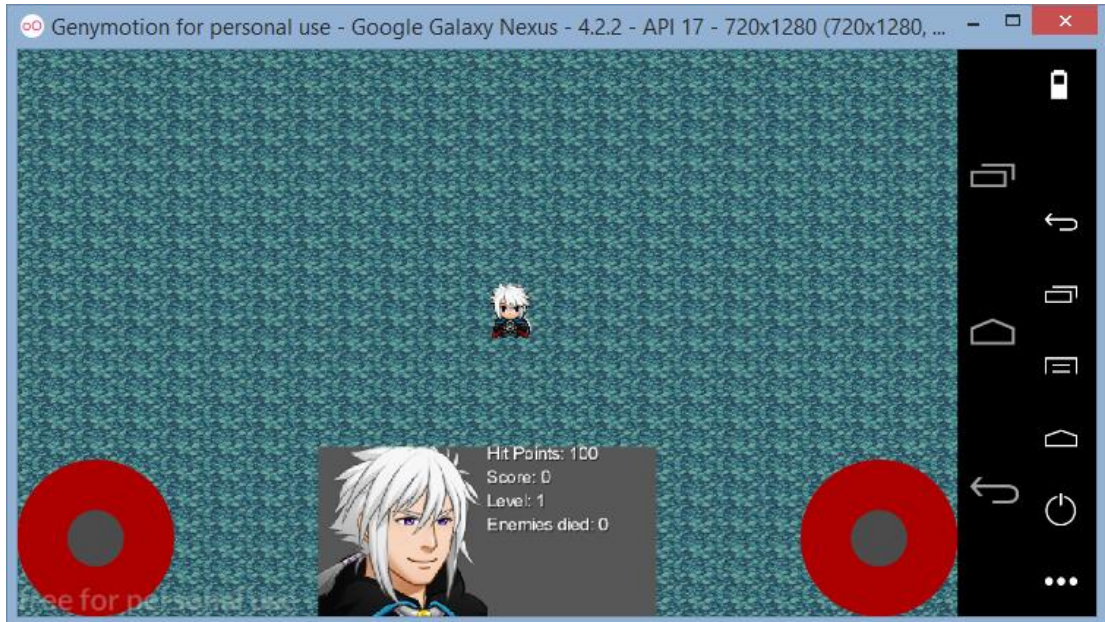


Рис. Гра «Siege», на емуляторі смартфоні

Ігровий процес

Для того щоб переходити на наступні рівні необхідно перемогти певну кількість ворогів.

Вороги йтимуть до героя з певною швидкістю й коли вони зіткнуться, герой отримує пошкодження та переміститься у довільну частину екрану. Коли кількість здоров'я героя дійде до 0, гра покаже екран програшу та вимкнеться.

Коли кількість здоров'я ворогів дійде до 0, вони зникають, а нові з'являються за межами екрану, головний герой отримує n-ну кількість балів, яка дорівнює вартості переможеного ворога.

Гра завершиться, коли герой переможе боса та потрапить у Мармуровий палац (а саме у середину верхньої частини екрану).